

DB2 10 for z/OS Performance Topics

Discover the functions that provide reduced CPU time in CM and NFM

Understand improved scalability and availability

Evaluate the impact of new functions



Paolo Bruni
Felipe Bortoletto
Ravikumar Kalyanasundaram
Glenn McGeoch
Roger Miller
Cristian Molaro
Yasuhiro Ohmori
Michael Parbs

Redbooks



International Technical Support Organization

DB2 10 for z/OS Performance Topics

June 2011

Note: Before using this information and the product it supports, read the information in “Notices” on page xxi.

First Edition (June 2011)

This edition applies to IBM DB2 Version 10.1 for z/OS (program number 5605-DB2).

© Copyright International Business Machines Corporation 2011. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	ix
Tables	xv
Examples	xvii
Notices	xxi
Trademarks	xxii
Preface	xxiii
The team who wrote this book	xxiii
Now you can become a published author, too!	xxvi
Comments welcome	xxvi
Stay connected to IBM Redbooks	xxvii
Summary of changes	xxix
June 2011, First Edition	xxix
December 2011, First Update	xxix
January 2013, Second Update	xxix
Chapter 1. Introduction	1
1.1 Executive summary	2
1.1.1 Performance benefit summary	2
1.1.2 Conclusion	2
1.2 General introduction to DB2 10	2
1.2.1 Performance improvements	2
1.2.2 Unsurpassed resiliency for business-critical information	3
1.2.3 Rapid application and warehouse deployment for business growth	4
1.2.4 Enhanced business analytics and mathematical functions with QMF	5
1.3 Performance expectations for DB2 10	5
1.3.1 Insert performance	8
1.3.2 When is it necessary to REBIND	9
1.3.3 What else is needed to get performance out-of-the-box	10
1.3.4 DB2 10 improvements for RELEASE(DEALLOCATE)	11
1.3.5 Performance estimation	11
1.4 How to read this book	11
Chapter 2. Subsystem	15
2.1 Catalog restructure	16
2.1.1 Catalog changes	17
2.1.2 Impact of DB2 catalog migration	18
2.1.3 DDL performance and concurrency	19
2.1.4 BIND and REBIND stability and performance	19
2.1.5 Compression and inline LOBs for SPT01	21
2.2 Latching contention relief	22
2.2.1 Latch class 19	23
2.2.2 Latch class 24 (EDM)	23
2.2.3 Latch class 32	25
2.2.4 UTSERIAL elimination	25
2.3 Dynamic prefetch enhancements	26

2.3.1 Disorganized index scan using list prefetch	27
2.3.2 Row level sequential detection	28
2.3.3 Progressive prefetch quantity	30
2.3.4 Summary on prefetch improvements	30
2.4 Buffer pool enhancements	30
2.4.1 Buffer storage allocation	31
2.4.2 In-memory table spaces and indexes	31
2.4.3 DB2 10 buffer pool prefetch and deferred write activities	33
2.5 Work file enhancements	34
2.5.1 Support for spanned work file records	34
2.5.2 In-memory work file enhancements	34
2.5.3 Work file table spaces	37
2.6 Logging enhancements	39
2.6.1 Log latch contention reduction	39
2.6.2 Long term page fix log buffers	39
2.6.3 LOG I/O enhancements	40
2.6.4 Performance with log writes	41
2.7 I/O parallelism for index updates	43
2.8 Space search improvement	45
2.9 Log record sequence number spin avoidance for inserts to the same page	46
2.10 Compression on insert	46
Chapter 3. Synergy with z platform	49
3.1 1 MB page frame support	50
3.2 zIIP usage with DB2 10	50
3.2.1 RUNSTATS zIIP eligibility	51
3.2.2 Asynchronous I/O zIIP eligibility	56
3.3 Open and close data sets	63
3.4 Disk storage enhancements	65
3.4.1 Prefetch improvement through disk enhancement	65
3.4.2 DB2 logging and insert with disk enhancements	68
3.4.3 Utilities and storage enhancement	69
3.4.4 DB2 support for solid state drives	73
3.5 SMF compression	74
Chapter 4. Table space design options	77
4.1 Universal table space	78
4.1.1 The use of UTS in DB2 9	78
4.1.2 The use of UTS in DB2 10	78
4.1.3 How to convert to UTS	79
4.1.4 New default table space at CREATE time	80
4.1.5 MEMBER CLUSTER option available for UTS	80
4.1.6 UTS workload performance	81
4.1.7 Summary for universal table spaces	85
4.2 XML	85
4.2.1 XML transaction processing performance	86
4.2.2 Modifying part of an XML document	87
4.2.3 Indexes on XML DATE and TIMESTAMP data	91
4.2.4 XML schema validation	92
4.2.5 XML type modifier	93
4.2.6 Support for binary XML	94
4.2.7 Support for multiple versions of XML documents	95
4.3 Inline LOBs	95

4.3.1	Advantages of inline LOBS	96
4.3.2	Inline LOBs performance	98
4.3.3	Queries for LOB size distribution	109
4.3.4	Inline LOBs: Conclusions	111
4.4	Hash access	112
4.4.1	Choosing hash table candidates	112
4.4.2	Hash overflow area	113
4.4.3	Converting to hash tables	114
4.4.4	SQL access performance	115
4.4.5	DDL and utilities	118
4.4.6	Monitoring the performance of hash access tables	120
Chapter 5.	Sample workloads	123
5.1	OLTP workloads	124
5.1.1	DB2 10 and SAP on IBM System z	124
5.1.2	IRWW workload	127
5.2	Virtual and real storage	134
5.2.1	Common storage and real storage	136
5.2.2	Subsystems consolidation	140
5.2.3	Storage use measurements	141
5.2.4	SAP workload	143
5.3	INSERT performance improvements	148
5.3.1	Insert performance summary	148
5.3.2	Insert performance measurements	150
Chapter 6.	SQL	153
6.1	IN-list enhancements	154
6.1.1	Matching multiple IN-list predicates	154
6.1.2	Avoiding additional index probing overhead	155
6.1.3	IN-list predicate transitive closure	156
6.1.4	List prefetch access for IN-list	157
6.2	Range-list index scan	157
6.3	Parallelism enhancements	159
6.3.1	Record range partitioning	160
6.3.2	Straw model for workload distribution	162
6.3.3	Sort merge join improvements	164
6.3.4	Removal of some parallelism restrictions	164
6.3.5	Query parallelism degree change	165
6.3.6	Parallelism enhancements performance summary	166
6.4	Predicate processing enhancements	166
6.4.1	Predicate evaluation enhancement	167
6.4.2	Residual predicate enhancements	172
6.5	Index probing	176
6.6	RID list work file overflow	179
6.7	Aggressive merge for views and table expressions	181
6.7.1	Correlated table expression	181
6.7.2	Table expression on preserved side of outer join	183
6.8	Implicit casting extension	185
Chapter 7.	Application enablement	187
7.1	Temporal support	188
7.1.1	New table attributes	188
7.1.2	Data versioning	189
7.1.3	Application performance comparisons	190

7.1.4 Productivity improvements with temporal feature	201
7.1.5 Improved data warehousing capabilities	201
7.1.6 Summary on temporal support	201
7.2 Referential integrity checking improvements	202
7.2.1 Avoiding checking for existing foreign key values	203
7.2.2 Exploiting index look-aside	204
7.2.3 Batch insert with referential integrity	205
7.2.4 Summary on referential integrity	206
7.3 Support for TIMESTAMP WITH TIMEZONE	206
7.4 Additional non-key columns in a unique index	208
7.5 Dynamic SQL literal replacement	212
7.6 EXPLAIN MODE special register to explain dynamic SQL	218
7.7 Access plan stability	219
7.7.1 EXPLAIN PACKAGE	220
7.7.2 APCOMPARE and APREUSE BIND options	221
7.8 Access currently committed data	230
7.8.1 Overview	230
7.8.2 Measurements	231
Chapter 8. Distributed environment	239
8.1 High performance DBATs	240
8.1.1 High Performance DBATs and RELEASE(DEALLOCATE)	241
8.1.2 High performance DBAT and RELEASE(DEALLOCATE) performance	246
8.2 Limited block fetch extended to the JCC Type 2 drivers	248
8.2.1 Large result set	250
8.2.2 Single row result set	252
8.2.3 IRWW workload	253
8.2.4 Limited block fetch summary	254
8.3 Return to client result sets	254
8.3.1 Test scenarios	255
8.3.2 Test results	255
8.4 Enhanced support for native SQL procedures	260
8.5 Extended correlation token	260
8.6 Virtual and real storage with distributed IRWW workload	262
8.7 LOBs and XML materialization avoidance	265
Chapter 9. Utilities	269
9.1 Use of FlashCopy in utilities	270
9.1.1 COPY utility	271
9.1.2 RECOVER utility	274
9.2 RUNSTATS	275
9.3 RECOVER with BACKOUT YES	279
9.4 Online REORG enhancements	279
9.4.1 REORG for base tables spaces with LOBs	279
9.4.2 Online REORG and prefetch	281
9.5 Increased availability for CHECK utilities	283
9.6 REPORT utility output improvement	283
9.7 Utility BSAM enhancements for extended format data sets	284
9.8 LOAD and UNLOAD	284
9.8.1 LOAD and UNLOAD with spanned records	284
9.8.2 LOAD and UNLOAD internal format	286
9.8.3 LOAD PRESORTED	287
9.9 DFSORT	288

9.9.1 DFSORT additional zIIP redirect	288
9.9.2 DFSORT performance enhancements	289
Chapter 10. Security	291
10.1 Policy-based audit capability	292
10.1.1 Audit policies	292
10.1.2 Benefits of DB2 10 audit policies	298
10.2 Support for row and column access control	299
10.2.1 Row permission performance	299
10.2.2 Column mask performance	300
10.3 Recent maintenance notes	301
Chapter 11. Installation and migration	303
11.1 Before you start	304
11.2 Installation	304
11.3 Migration	305
11.3.1 Introduction to migration to DB2 10	305
11.3.2 Summary of catalog changes	308
11.3.3 Catalog migration	309
11.3.4 Rebind during migration	310
11.3.5 Migration steps and performance	310
11.3.6 Conclusions and considerations	320
Chapter 12. Monitoring and Extended Insight	323
12.1 DB2 10 enhanced instrumentation	324
12.1.1 One minute statistics trace interval	324
12.1.2 IFCID 359 for index leaf page split	324
12.1.3 Separate DB2 latch and transaction lock waits in Accounting class 8	324
12.1.4 Storage statistics for DIST address space	325
12.1.5 Accounting: zAAP and zIIP SECP values	327
12.1.6 Package accounting information with rollup	328
12.1.7 DRDA remote location statistics detail	329
12.2 Enhanced monitoring support	330
12.2.1 Unique statement identifier	331
12.2.2 New monitor class 29 for statement detail level monitoring	332
12.2.3 System level monitoring	333
12.3 OMEGAMON PE Extended Insight	340
12.3.1 Examples	341
12.3.2 Configuring a CLI application	348
Appendix A. Recent maintenance	353
A.1 DB2 APARs	354
A.2 z/OS APARs	362
A.3 OMEGAMON PE APARs	363
Abbreviations and acronyms	365
Related publications	369
IBM Redbooks publications	369
Other publications	369
Online resources	370
Help from IBM	371
Index	373

Figures

1-1 CPU reduction across some workloads	6
1-2 The CPU performance measurements from the beta program.....	7
1-3 Impact of binding on IRWW workload.....	9
2-1 DB2 catalog evolution.....	17
2-2 Reduction on latch class 24	24
2-3 Disorganized index scan.....	27
2-4 Total elapsed time and dynamic prefetch I/O versus cluster ratio	29
2-5 Row level sequential detection with larger row.....	29
2-6 ALTER BUFFER POOL PGSTEAL syntax.....	33
2-7 DB2 installation panel DSNTIPB showing option 10 WFDBSEP	38
2-8 DB2 9 COMMIT synchronous I/O	40
2-9 DB2 10 COMMIT synchronous I/O.....	41
2-10 Suspension time per commit.....	42
2-11 Suspension time per commit.....	42
2-12 Maximum DB2 log throughput	43
2-13 I/O parallelism for index updates	44
2-14 Insert index I/O parallelism	45
2-15 Message DSNU241I compression dictionary built	47
3-1 RUNSTATS zIIP redirection eligibility	52
3-2 RUNSTATS elapsed time with one zIIP	52
3-3 DB2 RUNSTATS utility accounting report	53
3-4 WLM classification for batch job	54
3-5 RMF workload activity report for the RUNSTATS report class	55
3-6 RMF workload activity report for the DBM1 address space report class	56
3-7 OMEGAMON PE statistics trace sample	56
3-8 DB2 accounting and DB2 statistics.....	57
3-9 DB2 Data Studio, access path indicating sequential prefetch	58
3-10 Open 100,000 data sets using 20 concurrent jobs	64
3-11 FICON versus zHPF for 4 KB page sequential prefetch.....	66
3-12 FICON versus zHPF for 4 KB page dynamic prefetch	66
3-13 The 4 KB page prefetch, read from disk using zHPF	67
3-14 Sequential read from DS8800 disk by DB2	67
3-15 Maximum DB2 log throughput	68
3-16 Load preformat	69
3-17 LOAD utility measurement results	70
3-18 LOAD REPLACE of LOBs	70
3-19 UNLOAD I/O throughput measurement	71
3-20 The EF BSAM measurements	72
3-21 BSAM read changing the number of streams.....	72
3-22 BSAM writes changing number of stream.....	73
3-23 Tracing parameters panel DSNTIPN	74
3-24 Sample DSNTSMPD output	75
4-1 Possible table space type conversions	79
4-2 Workload environment definition.....	82
4-3 Non-range defined table spaces.....	83
4-4 Range defined table spaces	84
4-5 XMLMODIFY performance measurements for small documents	89
4-6 XMLMODIFY performance measurements for medium documents	90

4-7 XMLMODIFY performance measurements for large documents	91
4-8 XML validation measurement results	93
4-9 Binary XML measurement results	94
4-10 DASD space used for 1 million LOBs with 4 KB LOB page size.	98
4-11 Class 2 elapsed time to select 10,000 x 200 byte LOBs.	99
4-12 Class 2 elapsed time to insert 10,000 x 200 byte LOBs	100
4-13 Class 2 CPU time to insert 10,000 x 200 byte LOBs	100
4-14 Class 2 elapsed time for 5,000 random updates/deletes of 200 byte LOBs	101
4-15 Class 2 CPU time for 5,000 random updates/deletes of 200 byte LOBs	101
4-16 Class 1 elapsed time for LOAD REPLACE of small LOBs	102
4-17 Class 1 CPU time for LOAD REPLACE of small LOBs.	103
4-18 Class 1 elapsed time for UNLOAD of small LOBs	103
4-19 Class 1 CPU time for UNLOAD of small LOBs	104
4-20 Class 1 elapsed time for create spatial index on inline LOB column	104
4-21 Class 1 elapsed time for spatial queries using spatial indexes.	105
4-22 Tuning inline LOBs - Cumulative LOB size distribution - test cases 1 to 4	106
4-23 Tuning inline LOBs - Cumulative LOB size distribution - test cases 5 to 8	107
4-24 Tuning inline LOBs - Cumulative LOB size distribution - test cases 9 to 11	108
4-25 Access to hash table.	114
4-26 Hash table - Overview process of converting and creating.	115
4-27 Select hash table versus Select 3-level indexed table	116
4-28 Select hash table versus Select 3-level indexed access only.	116
4-29 Update hash table versus Update 3-level indexed table.	117
4-30 Insert/Delete Hash Table versus Insert/Delete 3-level indexed table	117
4-31 IRWW workload with hash tables	118
5-1 DB2 9 and DB2 10 ITR using SAP SD	125
5-2 DB2 9 and DB2 10 N-way Scaling with SAP SD	126
5-3 IRWW OLTP non-data sharing measurements.	128
5-4 IRWW OLTP data sharing measurements, RELEASE(COMMIT)	128
5-5 IRWW OLTP data sharing measurements, RELEASE(DEALLOCATE)	129
5-6 IRWW OLTP Non DS and DS, DBM1 and MVS storage below 2 GB bar	129
5-7 Total CPU time, DB2 9 versus DB2 10 compared for distributed IRWW	131
5-8 DB2 10 distributed stored procedures, CPU and PKGREL	133
5-9 DB2 10 Distributed Workloads, CPU and PKGREL	133
5-10 DBM1 address space memory relief across versions.	135
5-11 Layout of z/OS virtual storage.	136
5-12 DB2 running large number of threads.	140
5-13 DBM1 storage below-the-bar for our dynamic SQL workload.	142
5-14 DBM1 Storage below-the-bar for our static SQL workload.	143
5-15 DB2 9 and DB 10 virtual storage usage with SAP SD	144
5-16 Measurement data - 2,500 threads with DB2 10	145
5-17 DB2 10 ITR and response time when vary MAXKEEP with SAP SD	146
5-18 DB2 10 virtual storage when varying MAXKEEP with SAP SD	147
5-19 Summary of main insert performance improvements	148
5-20 Sequential insert performance improvement	150
5-21 Middle-sequential inserts	151
5-22 Sequential INSERT.	151
5-23 Random INSERT	152
6-1 Explain for matching multiple IN-list predicates	154
6-2 Explain for IN-list predicate transitive closure	156
6-3 Explain for list prefetch usage for IN-list access	157
6-4 Explain for range-list index scan	158
6-5 Explain for key range partitioning in DB2 9.	161

6-6 Explain for DB2 10 record range partitioning	161
6-7 Non-straw model versus straw model.	163
6-8 Predicate evaluation savings for index access	168
6-9 Predicate evaluation savings for table space scan access.	169
6-10 Machine code generation savings for IN-list with no qualifying rows	170
6-11 Machine code generation savings for LIKE with no qualifying rows	171
6-12 Residual predicate enhancements diagram	172
6-13 Residual predicate pushdown savings for index access.	174
6-14 Residual predicate pushdown savings for table space scan access	174
6-15 Residual predicate pushdown savings for varying number of qualifying rows	176
6-16 Explain results for index probing (non VOLATILE)	178
6-17 Explain results for index probing (VOLATILE)	179
6-18 RID pool overflow performance numbers	180
6-19 Explain for DB2 9 correlated table expression query	182
6-20 Explain for DB2 10 correlated table expression query	182
6-21 Performance for merge of correlated table expression.	183
6-22 Explain for DB2 9 table expression on preserved outer join query	184
6-23 Explain for DB2 10 table expression on preserved outer join query	184
6-24 Performance for merge of table expression on preserved side of outer join.	185
7-1 DB2 system time temporal versus user defined trigger solution for a mixed workload	191
7-2 UPDATE performance with SYSTEM TIME temporal versus trigger solution.	193
7-3 DELETE performance with SYSTEM TIME temporal versus trigger solution.	194
7-4 UPDATE performance on business time temporal versus stored procedure solution	197
7-5 INSERT performance WITH versus WITHOUT OVERLAPS index	197
7-6 System Time Temporal SELECT statement#1 - Access path	199
7-7 System Time Temporal SELECT statement#2 - Access path	199
7-8 System Time Temporal SELECT statement#3 - Access path	200
7-9 System Time Temporal SELECT statement#4 - Access path (base table only).	200
7-10 Index getpage reduction from avoiding referential integrity checking	204
7-11 Index getpage reduction from index look-aside	205
7-12 DB2 elapsed and CPU time comparison for referential integrity checking enhancements	206
7-13 Index definitions for additional non-key index columns tests	209
7-14 INSERT performance measurements for additional non-key index columns	209
7-15 Performance measurements for dynamic SQL literal replacement.	214
7-16 Performance measurements for dynamic SQL literal replacement - PREPARE only	215
7-17 Analysis of DSN_STATEMENT_CACHE_TABLE and LITERAL_REPL	218
7-18 EXPLAIN PACKAGE syntax	220
7-19 PREPARE clauses	230
7-20 Class 1 times for skip uncommitted inserts - Row level locking	232
7-21 Class 2 times for skip uncommitted inserts - Row level locking	233
7-22 Class 1 and 2 CPU times - Currently committed versus WITH UR.	235
7-23 CPU times for SELECT unblocked by DELETE - Row level locking.	235
7-24 Elapsed time for SELECT unblocked by DELETE - Row level locking.	236
7-25 CPU times for SELECT unblocked DELETE - Page level locking	237
8-1 MODIFY DDF PKGREL syntax.	242
8-2 DB2 Configuration Assistant, Bind panel	245
8-3 DB2 Configuration Assistant, Add Bind Option panel	246
8-4 The DB2 10 distributed application: RELEASE(COMMIT) versus RELEASE(DEALLOCATE).	247
8-5 JDBC T2 driver in DB2 9 for z/OS	249
8-6 JDBC T2 driver in DB2 10 for z/OS	250
8-7 Limited block fetch: Large result set	250

8-8	Limited block fetch overhead for single row result set	253
8-9	Limited block fetch: IRWW OLTP workload	253
8-10	Class 1 CPU measurements for WITH RETURN TO CLIENT - Single row result set	256
8-11	Class 2 CPU measurements for WITH RETURN TO CLIENT - Single row result set	257
8-12	Class 1 CPU measurements for WITH RETURN TO CLIENT - 500 row result set .	258
8-13	Class 2 CPU measurements for WITH RETURN TO CLIENT - 500 row result set .	259
8-14	DBM1 storage below the bar, distributed workload 9 versus 10	262
8-15	Distributed workload: DB2 9 versus DB2 10 total real storage utilization compared	263
8-16	Streaming LOBs and XML	266
8-17	Streaming effects with DRDA LOB INSERTs	267
9-1	COPY with FLASHCOPY NO accounting report.	272
9-2	COPY with FLASHCOPY YES accounting report.	272
9-3	Copy CPU time with FlashCopy Yes and No	273
9-4	Copy elapsed time with FlashCopy Yes and No.	273
9-5	RECOVER from FlashCopy	274
9-6	RUNSTATS sampling syntax	275
9-7	Sampling: Page-level versus row-level	276
9-8	Complex RUNSTATS (COLGROUP and HISTOGRAM)	277
9-9	Basic RUNSTATS - Sampling 20%	277
9-10	Basic RUNSTATS - Sampling 10%	278
9-11	Basic RUNSTATS - Sampling 5%	278
9-12	REORG LOB AUX YES	280
9-13	UNLOAD LOB using spanned records	285
9-14	VBS versus USS.	286
9-15	LOAD and UNLOAD internal format	286
9-16	LOAD and UNLOAD internal format with compress	287
9-17	DFSORT V1R10 PM18196 and no zIIP	289
9-18	DFSORT V1R10 PM18196 and 1 zIIP	290
10-1	DSNTSMFD output for audit records - IFCID 144.	296
10-2	Audit policies performance	297
10-3	ITR with row permissions	300
11-1	DB2 version summary	304
11-2	Migration paths and modes.	306
11-3	CATMAINT elapsed and CPU time comparison.	311
11-4	CATMAINT execution for the three catalogs	312
11-5	CATMAINT elapsed and CPU time comparing with previous releases	312
11-6	CATENFM elapsed and CPU time comparison	313
11-7	CATENFM execution for the three catalogs	314
11-8	The DSNTIJEN elapsed and CPU time comparison with previous releases	314
11-9	DSNTIJEN measurement details	315
11-10	CATENFM elapsed time and CPU time changing BP0 size	316
11-11	CATENFM elapsed and CPU time changing BP8K0 size.	317
11-12	Skip level migration - Elapsed time.	318
11-13	Skip level migration - CPU time	318
11-14	Migration elapsed time - 2-way data sharing	319
11-15	Migration CPU time - 2-way data sharing	320
12-1	Accounting suspend times	325
12-2	Statistics, DIST storage above 2 GB	327
12-3	Statistics, DIST storage below 2 GB.	327
12-4	Performance impact of monitoring class 29	332
12-5	Performance impact of performance IFCIDs.	333
12-6	System level monitoring - Invalid profile	335
12-7	System level monitoring - Attributes table.	338

12-8 Components of an End-to-End response time	341
12-9 Overview of Optim Performance Expert Extended Insight architecture	342
12-10 OMEGAMON PE EI Analysis Dashboard	343
12-11 OMEGAMON Performance Manager main screen	344
12-12 Adding a database for monitoring to OMEGAMON PE EI	344
12-13 OMEGAMON PE EI - configuring a database	345
12-14 OMEGAMON PE EI configure monitoring	346
12-15 Review configuration settings for OMEGAMON PE EI	347
12-16 OMEGAMON PE EI Client configuration tool	348
12-17 Indicating the db2dsdriver.cfg file to update for OMEGAMON PE EI	349
12-18 Configuration of a CLI application for OMEGAMON PE EI	350

Tables

2-1 Test scenario for prefetch performance reports - Row size 49 bytes	28
2-2 Test scenario for prefetch performance reports - Row size 98 bytes	29
4-1 Cumulative LOB size % distribution - test cases 9 to 11.	108
4-2 Performance of CREATE a hash table.	119
4-3 Loading a large hash table	120
5-1 SAP Sales and Distribution (SD) workload measurements	124
5-2 Classic versus UTS table space measurements for non-data sharing	130
5-3 RELEASE(COMMIT) versus RELEASE(DEALLOCATE) for distributed applications	132
5-4 Virtual storage relief allows system parameters with new maximums	141
5-5 DBM1 Storage below-the-bar for our dynamic SQL workload - numbers	142
5-6 DBM1 Storage below-the-bar for our static SQL workload - numbers	143
5-7 Concurrent threads on DB2 9 and DB2 10 with SAP SD	144
6-1 Matching multiple IN-list predicate performance numbers	155
6-2 Range-list index scan performance numbers	159
6-3 Record range partitioning performance numbers	162
6-4 Multi-row fetch parallelism performance numbers	165
6-5 Distribution of column values for index probing test	177
6-6 Implicit cast target data types and length	186
7-1 System time temporal versus user defined trigger solution on a mixed workload	190
7-2 Temporal queries with explicit and implicit UNION ALL with history tables	198
7-3 Productivity improvements using temporal capability	201
7-4 Performance measurements for TIMESTAMP WITH TIMEZONE	207
7-5 Query performance impact for additional non-key columns in unique index.	211
7-6 Class 2 CPU time for dynamic SQL literal replacement tests	215
7-7 Class 2 CPU time for dynamic SQL literal replacement tests - PREPARE only	216
7-8 PLAN_TABLE row for EXPLAIN PACKAGE statement	221
7-9 Distribution of data in table for APCOMPARE and APREUSE examples.	223
7-10 PLAN_TABLE contents for initial BIND of program APCMPCOB.	224
7-11 PLAN_TABLE contents for initial BIND and for BIND with APCOMPARE(ERROR)	225
7-12 PLAN_TABLE contents for REBIND with APREUSE - access path is reused	228
7-13 Workload descriptions for access currently committed data tests	231
7-14 Class 1 and 2 times for skip uncommitted inserts - Page level locking.	233
7-15 Class 1 and 2 times for SKIPUNCI versus new feature - Row level locking.	234
7-16 Elapsed times for select unblocked delete - Page level locking	237
7-17 Class 1 and 2 times - Wait for commit versus skip uncommitted INSERT	238
8-1 RELEASE(COMMIT) versus RELEASE(DEALLOCATE) for distributed applications	248
8-2 Default, minimum, and maximum values of queryDataSize	251
9-1 TABLESAMPLE options	276
9-2 Performance of REORG TABLESPACE PART SHRLEVEL(CHANGE).	281
9-3 Performance of REORG TABLESPACE PART SHRLEVEL(CHANGE) detail	281
9-4 Performance of NPI - REORG INDEX SHRLEVEL(CHANGE)	282
9-5 Performance of NPI - REORG INDEX SHRLEVEL(CHANGE) details	283
9-6 LOAD PRESORTED YES option	287
10-1 Mapping of audit class to audit policies	293
10-2 Measured values in seconds	296
11-1 Details of buffer pool statistics	316
A-1 DB2 10 current function and performance related APARs	354
A-2 z/OS DB2-related APARs	362
A-3 OMEGAMON PE GA and DB2 10 related APARs	363

Examples

2-1 Sample query for GROUP BY performance measurements.....	36
2-2 OMEGAMON PE syntax for reporting IFCIDs 95 and 96.....	36
2-3 OMEGAMON PE record trace report for IFCIDs 95 and 96.....	36
3-1 JCL RMF workload activity report.....	54
3-2 Simple query example, table space scan.....	58
3-3 Modify Trace JCL example	59
3-4 Modify Trace execution example	59
3-5 OMEGAMON PE JCL for reporting DB2 prefetch.....	59
3-6 OMEGAMON PE Accounting: sequential prefetch	60
3-7 OMEGAMON PE accounting: special engine CPU time.....	60
3-8 OMEGAMON PE statistics trace.....	61
3-9 WLM classification of DB2 address spaces into reporting classes.....	62
3-10 RMF post processor syntax example	62
3-11 RMF Workload Activity report	63
4-1 Sample use of XMLMODIFY function for sub-document UPDATE.....	87
4-2 UPDATE statements using XMLMODIFY to test partial update - part 1.....	87
4-3 UPDATE statements using XMLMODIFY to test partial update - part 2.....	88
4-4 Sample use of XMLMODIFY that is poorly performing.....	90
4-5 Sample DDL to create index on XML data using DATE data type	92
4-6 Sample query to produce LOB column size distribution	109
4-7 Report produced from LOB column size cumulative distribution query	110
4-8 Building the query in Example 4-6 for all LOBs in tables created by a TBCREATOR	110
5-1 DBM1 storage below the 2 GB bar information	137
5-2 DBM1 above the 2 GB bar storage layout	138
5-3 Distributed address space storage below and above the 2 GB bar	139
5-4 Real and auxiliary storage information for DBM1 and DIST address spaces	139
5-5 Common and subsystem shared storage report.....	139
6-1 Matching multiple IN-list predicates	154
6-2 Avoid additional index probing overhead	155
6-3 Predicate transitive closure for IN-lists	156
6-4 List prefetch for IN-list predicates	157
6-5 Range-list index scan	158
6-6 Record range partitioning query	160
6-7 Parallelism with multi-row fetch.....	165
6-8 Predicate evaluation enhancements - 3 predicate test.....	167
6-9 Predicate evaluation enhancements - 10 predicate test.....	167
6-10 Machine code generation enhancement - 100 item IN-list test.....	169
6-11 Machine code generation enhancement - 50 item IN-list test.....	170
6-12 Machine code generation enhancement - 10 item IN-list test.....	170
6-13 Query with three predicates for residual pushdown test.....	173
6-14 Query with ten predicates for residual pushdown test	173
6-15 WHERE clauses for residual predicate pushdown qualifying rows tests	175
6-16 INSERT statements for index probing test case	177
6-17 Aggressive merge for correlated table expression	181
6-18 Aggressive merge for table expression on preserved side of outer join	183
7-1 DDL code to enable versioning of data - System Time temporal capability of DB2 ..	189
7-2 SQL statements used for trigger solution on a regular table UPDATE	191
7-3 SQL statements used for system time temporal table UPDATE	192

7-4	SQL statements used for trigger solution on a regular table DELETE	193
7-5	SQL statements used for System time temporal table DELETE.....	194
7-6	UPDATE statement on business time temporal table.....	195
7-7	SQL for UPDATE performance on a regular table with stored procedure solution....	195
7-8	Sample SQL accessing base (temporal) table - SQL statement#4.....	200
7-9	Table with time zone data.....	207
7-10	Multiple unique indexes in DB2 9 for additional non-key column query test.....	210
7-11	Single unique index in DB2 10 with additional non-key columns	210
7-12	Query to test impact of additional non-key columns in unique index	211
7-13	Dynamic SQL literal replacement	213
7-14	Sample query for testing CSWL performance improvement.....	214
7-15	OMEGAMON PE report showing statistics on CSWL.....	216
7-16	OMEGAMON PE accounting command example for CSWL	217
7-17	EXPLAIN STMTCACHE ALL	217
7-18	Sample EXPLAIN PACKAGE statement	221
7-19	DDL to create table and index for APCOMPARE and APREUSE examples	223
7-20	SELECT statement for use in APCOMPARE and APREUSE examples	224
7-21	DDL to create second index for APCOMPARE and APREUSE examples.....	224
7-22	BIND options to only compare access paths and see what has changed	225
7-23	Output from REBIND using APCOMPARE(ERROR) and EXPLAIN(ONLY)	225
7-24	BIND options to use APCOMPARE to fail REBIND if access path has changed	226
7-25	Output from REBIND using APCOMPARE(ERROR) and EXPLAIN(YES).....	226
7-26	BIND options to use APCOMPARE to warn us if access path has changed	227
7-27	Output from REBIND using APCOMPARE(WARN) and EXPLAIN(YES).....	227
7-28	BIND options to use APREUSE to rebind using the existing access path	228
7-29	Output from REBIND using APREUSE where access path is reused	228
8-1	-DIS DDF command reporting the PKGREL option	241
8-2	MODIFY DDF PKGREL(BNDOPT) output	242
8-3	MODIFY DDF PKGREL(COMMIT) output	242
8-4	BIND COPY command	243
8-5	DB2 Administration Tool view of packages.....	244
8-6	SET CURRENT PACKAGE PATH	244
8-7	db2sqljcustomize and -collection parameter.....	245
8-8	Observing in the DB2 accounting if limited block fetch is active.....	251
8-9	Accounting for single row result set with and without limited block fetch	252
8-10	Chaining SET statements	260
8-11	DB2 9 for z/OS report of a distributed application being TIMED OUT	261
8-12	DB2 10 for z/OS report showing extended correlation token information.....	261
8-13	OMEGAMON PE RETRACE report syntax	263
8-14	Reporting DIST address space storage with IFCID 225.....	263
8-15	OMEGAMON PE Statistics report syntax	264
8-16	Reporting DIST address space storage in OMEGAMON PE Statistics Report Long	264
10-1	START TRACE for auditing DML access to a table with audit trace.....	294
10-2	OMEGAMON IFCID frequency distribution log for audit trace with static SQL statements	294
10-3	Audit policy creation	294
10-4	START TRACE for auditing DML access to a table with audit policies.....	295
10-5	OMEGAMON IFCID frequency distribution log for audit policies with static SQL statements	295
10-6	IFCID counts with audit policies turned off	297
10-7	IFCID counts with audit policies turned on	297
10-8	A SQL statement sample in loop	298
10-9	Two SQL statements in loop.....	298

10-10	Sample row permission for measurement	299
10-11	Column mask definition	300
10-12	View definition	301
11-1	Sample output from Display Group command	309
12-1	OMEGAMON PE RECTRACE report syntax	325
12-2	OMEGAMON PE record trace report example	326
12-3	Starting CLASS 29 traces	332
12-4	Starting performance traces	332
12-5	db2dsdriver.cfg example for OMEGAMON PE EI support	351

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®	IMS™	Resource Measurement Facility™
CICS®	MQSeries®	RETAIN®
DB2 Connect™	MVS™	RMF™
DB2®	OMEGAMON®	Service Request Manager®
developerWorks®	Optim™	System Storage®
DRDA®	OS/390®	System z10®
DS8000®	Parallel Sysplex®	System z9®
Enterprise Storage Server®	POWER6+™	System z®
Enterprise Workload Manager™	POWER6®	Tivoli®
FICON®	PR/SM™	WebSphere®
FlashCopy®	pureXML®	z/Architecture®
GDPS®	QMF™	z/OS®
Geographically Dispersed Parallel Sysplex™	Query Management Facility™	z10™
HiperSockets™	RACF®	z9®
IBM®	Redbooks®	zSeries®
	Redbooks (logo) 	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM DB2® 10 can reduce the total DB2 CPU demand from 5-20% when you take advantage of all the enhancements. Many CPU reductions are built in directly to DB2, requiring no application changes. Some enhancements are implemented through normal DB2 activities through rebinding, restructuring database definitions, improving applications, and utility processing. The CPU demand reduction features have the potential to provide significant total cost of ownership savings based on the application mix and transaction types.

Improvements in optimization reduce costs by processing SQL automatically with more efficient data access paths. Improvements through a range-list index scan access method, list prefetch for IN-list, more parallelism for select and index insert processing, better work file usage, better record identifier (RID) pool overflow management, improved sequential detection, faster log I/O, access path certainty evaluation for static SQL, and improved distributed data facility (DDF) transaction flow all provide more efficiency without changes to applications. These enhancements can reduce total CPU enterprise costs because of improved efficiency in the DB2 10 for IBM z/OS®.

DB2 10 includes numerous performance enhancements for Large Objects (LOBs) that save disk space for small LOBs and that provide dramatically better performance for LOB retrieval, inserts, load, and import/export using DB2 utilities. DB2 10 can also more effectively REORG partitions that contain LOBs.

This IBM® Redbooks® publication provides an overview of the performance impact of DB2 10 for z/OS discussing the overall performance and possible impacts when moving from version to version. We include performance measurements that were made in the laboratory and provide some estimates. Keep in mind that your results are likely to vary, as the conditions and work will differ.

In this book, we assume that you are somewhat familiar with DB2 10 for z/OS. See *DB2 10 for z/OS Technical Overview*, SG24-7892-00, for an introduction to the new functions.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Paolo Bruni is a DB2 Information Management Project Leader at the International Technical Support Organization based in the Silicon Valley Lab. He has authored several IBM Redbooks publications about DB2 for z/OS and related tools and has conducted workshops and seminars worldwide. During Paolo's many years with IBM, in development and in the field, his work has been mostly related to database systems.

Felipe Bortolotto is a Certified IBM IT Specialist in information management and an IBM Certified DBA for DB2 for z/OS V7, V8, and DB2 9. He has 16 years of experience in IT with 11 years of experience with DB2 for z/OS. He joined IBM 7 years ago and is currently a member of the IBM Integrated Technology Delivery in Brazil. He holds a degree in Computer Science from UNICAMP. Felipe co-authored *Securing and Auditing Data on DB2 for z/OS*, SG247720.

Ravikumar Kalyanasundaram is a Certified IT Specialist. He works as a Managing Consultant with IBM SWG. Ravi has over 17 years of experience with database technology. He provides DB2 performance consulting services for large customers on z/OS and Linux®, UNIX®, and Windows® (LUW) systems. He holds a Bachelor's degree in Electrical and Electronics Engineering and a Masters degree in Business Administration. Ravi is also a certified Database Administrator on DB2 V8 and DB2 9. His interests are DB2 performance tuning and disaster recovery.

Glenn McGeoch is a Senior DB2 Consultant for the IBM DB2 for z/OS Lab Services organization in the United States, working out of San Francisco, CA. He has 33 years of experience in the software industry, with 25 years of experience working with DB2 for z/OS. He holds a degree in Business Administration from the University of Massachusetts and an MBA from Rensselaer Polytechnic Institute. Glenn worked for 19 years as an IBM customer with a focus on IBM CICS® and DB2 application development, and has spent the last 14 years with IBM assisting DB2 customers. His areas of expertise include application design and performance, stored procedures and DB2 migration planning. He has presented to regional DB2 User Groups and to customers on various DB2 topics. Glenn co-authored *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083 and *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604.

Roger Miller is a DB2 for z/OS evangelist, strategist, architect, designer, developer, writer, service, and factotum who has worked on many facets of DB2, ranging from overall design issues to SQL, languages, installation, security, audit, standards, performance, concurrency, and availability. He has worked for a few decades on DB2 development, product design, and strategy and has contributed directly or indirectly to most of the IBM Redbooks publications published so far. He often helps customers to use the product, answers many questions, and presents frequently to user groups.

Cristian Molaro is an IBM Gold Consultant, an independent DB2 specialist, and an instructor based in Belgium. He was recognized by IBM as an Information Champion in 2009, 2010, and 2011. His main activity is linked to DB2 for z/OS administration and performance. Cristian is co-author of these IBM Redbooks publications: *Enterprise Data Warehousing with DB2 9 for z/OS*, SG24-7637, *50 TB Data Warehouse Benchmark on IBM System z*, SG24-7674, *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01, and *Co-locating Transactional and Data Warehouse Workloads on System z*, SG24-7726. He holds a Chemical Engineering degree and a Masters degree in Management Sciences. He can be reached at cristian@molaro.be.

Yasuhiro Ohmori is an Advisory IT Specialist with Technology & Solutions, Insurance No.1, IBM Japan, and also IBM Japan Systems Engineering Co., Ltd. (ISE) under GTS in Japan, providing technical support on DB2 for z/OS for Nippon Life Insurance Company or Nissay in Japan. He has 10 years of experience in technical support for DB2 for z/OS. Yasuhiro has worked with several major customers in Japan implementing DB2 for z/OS and has conducted workshops for IBMers in Japan. His areas of expertise include DB2 for z/OS, IBM DRDA® implementation, IBM DB2 Connect™, and related topics. Yasuhiro co-authored the IBM Redbooks publication, *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01.

Michael Parbs is a Senior DB2 Specialist with IBM Global Technology Services A/NZ, from Canberra, Australia. He has over 20 years experience with DB2, primarily on the z/OS platform. Before joining IBM he worked in the public sector in Australia, as a DB2 DBA and DB2 Systems Programmer. Since joining IBM, Michael has worked as a subject matter expert with a number of DB2 customers both in Australia and China. Michael's main areas of expertise are data sharing, and performance and tuning, but his skills include database administration and distributed processing. Michael is an IBM Certified IT Specialist in Data Management and has co-authored several IBM Redbooks publications, including *DB2 for MVS/ESA Version 4 Data Sharing Implementation*, SG24-4791, *DB2 UDB Server for OS/390 and z/OS Version 7 Presentation Guide*, SG24-6121, *DB2 UDB for z/OS Version 8*:

Everything You Ever Wanted to Know, ... and More, SG24-6079, *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465, and *DB2 10 for z/OS Technical Overview*, SG24-7892-00.

Very special thanks to Jeff Berger who contributed to this book in the form of advice, written content, and project support.

Special thanks to Catherine Cox and all the members of the DB2 Performance Department in SVL for providing the information necessary for this project.

Thanks to the following people for their contributions to this project:

Rich Conway
Bob Haimowitz
Emma Jacobs
International Technical Support Organization

Adarsh Pannu
Akiko Hoshikawa
Akira Shibamiya
Andy Lai
Binghua Zhen
Bob Lyle
Brian Baggett
Catherine Cox
Chung Wu
Dan Weis
David Dossantos
David Zhang
Frank Bower
Frank Butt
Frank Vitro
Gopal Krishnan
Irene Liu
Jae Lee
James Guo
Jason Cu
Jay Yothers
Jeff Berger
Kalpana Shyam
Keith Howell
Lingyun Wang
Mai Nguyen
Marko Dimitrijevic
Maryela Weihrauch
Meg Bernal
Neena Cherian
Nguyen Dao
Paramesh Desai
Rich Vivenza
Sueli Almeida
Terry Purcell
Todd Munk
Ying Chang
IBM Silicon Valley Lab

Miao Zheng
IBM China

Christian Michel
Norbert Heck
Norbert Jenninger
IBM Boeblingen Lab

Brenda Beane
Seewah Chan
Paul Lekkas
Howard Poole
Michael Sheets
IBM Poughkeepsie SAP Performance Evaluation on IBM System z®

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:
ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7942-00
for DB2 10 for z/OS Performance Topics
as created or updated on January 21, 2013.

June 2011, First Edition

The revisions of this First Edition, first published on June 11, 2011, reflect the changes and additions described below.

December 2011, First Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

Changed information

- ▶ Updated information on APARs in Appendix A, “Recent maintenance” on page 353.

New information

- ▶ Added information on APARs in Appendix A, “Recent maintenance” on page 353.

January 2013, Second Update

This revision reflects the addition, deletion, or modification of new and changed information described below.

Change bars reflect the updates.

Changed information

- ▶ Updated Figure 11-1 on page 304.
- ▶ Updated information on APARs in Appendix A, “Recent maintenance” on page 353.

New information

- ▶ Added information on APARs in Appendix A, “Recent maintenance” on page 353.



Introduction

Performance is enhanced with DB2 10 for z/OS, offering reduced regression and opportunities for reduced CPU time. For relational database customers, database performance is paramount. DB2 10 for z/OS can reduce CPU demand 5% to 10% immediately with no application changes. CPU demand can be further reduced up to 20% when using all the DB2 10 enhancements in new-function mode (NFM). By pushing the performance limits, IBM and DB2 10 continue to lead the database industry with state-of-the-art technology and the most efficient database processing available.

In this chapter, we discuss the following topics:

- ▶ Executive summary
- ▶ General introduction to DB2 10
- ▶ Performance expectations for DB2 10
- ▶ How to read this book

1.1 Executive summary

DB2 10 for z/OS provides a performance improvement for most customers and most workloads. This book provides details and performance measurements of the major new performance improvements. The measurements show broad ranges for benchmarks and for some customer examples.

1.1.1 Performance benefit summary

Many of the DB2 10 performance benefits deliver immediately with few database, application or system changes. Most require some system or database administration work. Some require careful tuning by database administrators after reaching new function mode. Some need changes in the client software or applications.

- ▶ CPU reductions:

DB2 enhancements improve application performance and reduce CPU usage. Most customers can expect to see net CPU savings of 5-10% in their traditional DB2 transaction and batch workloads when compared to DB2 9, without any application changes being required. Customers who move from DB2 V8 will get utility CPU reductions and additional improvements from insert work and optimization.

- ▶ Scalability improvements:

Much more concurrent work can run on DB2 10. Customers can have as many as 5 to 10 times the concurrent work. The scalability improvements allow many customers to consolidate their workloads and to use memory for better performance, resulting in net CPU and memory savings and improving application performance.

- ▶ Productivity enhancements:

New features such as temporal tables, automated statistics and improved dynamic schema change reduce the effort required by developers and support staff to deliver robust DB2 applications. Many of the CPU and scalability improvements help with productivity while they reduce administration and programmer work.

1.1.2 Conclusion

The DB2 10 improvements vary widely, depending upon the workload and the work done. Most customers will need to REBIND, monitor and tune. These performance improvements deliver real and quantifiable business benefit, and many customers will be considering upgrading to DB2 10 much more quickly than they might have done for previous releases. While you can get a better understanding of the improvements by understanding these measurements and comparing them to your situation, a reasonable estimation of what you will experience still requires that you measure the workload.

1.2 General introduction to DB2 10

In this section we discuss the various improvements provided by DB2 10.

1.2.1 Performance improvements

DB2 10 delivers by improving performance and reducing CPU usage. Most customers can achieve out-of-the-box CPU savings of 5% to 10% for traditional workloads and up to 20% for specific workloads described next. Measurements compare to previous releases of DB2 for z/OS. REBIND is needed to obtain the best performance and memory improvements.

DB2 reduces CPU usage by optimizing processor times and memory access, leveraging the latest processor improvements, larger amounts of memory, and z/OS enhancements. Improved scalability and constraint relief can add to the savings. Productivity improvements for database and systems administrators can drive even more savings.

In DB2 10, performance improvements focus on reducing CPU processing time without causing significant administration or application changes. Most performance improvements are implemented by simply migrating to DB2 10 and rebinding.

You gain significant performance improvements from distributed data facility (DDF) optimization, buffer pool enhancements, parallelism enhancements, and more.

Early DB2 10 performance benchmarking and customer experience have shown a 5% to 10% CPU reduction in transactions after rebinding. Some customers might get more and some less CPU reduction depending on the workload. Customers who have scalability issues, such as virtual storage constraints or latching, can see higher improvements. Opportunities for tuning can take advantage of memory improvements. More high volume, short-running distributed transactions can take advantage of CPU reductions, using the RELEASE(DEALLOCATE) bind option.

Concurrent sequential insert CPU time can be reduced from 5% to 40%. Queries can be improved as much as 20% without access path change, and more for better access paths. A native SQL procedure workload has shown up to 20% CPU reduction using SET statements, IF statements, and SYSDUMMY1. Customers moving from DB2 9 can expect to see no change in CPU times for utilities, whereas customers moving from DB2 V8 will see CPU reductions up to 20%.

Productivity improvements

New SQL and XML capabilities improve productivity for those who develop new applications and for those who are porting applications from other platforms. Automating, reducing, or eliminating tasks and avoiding manual actions improve productivity and can help avoid problems. Resiliency improvements for virtual storage and availability increase productivity. DB2 10 improvements make the install, migration, and service processes faster and more reliable. Installation and migration information has been improved, using customer feedback.

Flexibility in migration paths

For this release, you can upgrade to DB2 10 directly from a DB2 Version 8 subsystem in new-function mode without starting the system in DB2 9. This provides customers greater flexibility to meet their business needs and to save time getting to DB2 10. Several process improvements make the upgrade simpler.

1.2.2 Unsurpassed resiliency for business-critical information

Business resiliency is a key component of the value proposition of DB2 for z/OS, System z hardware, the z/OS operating system, and other key System z software, such as IBM IMS™ and CICS. Resiliency helps to keep your business running even during unexpected circumstances. Innovations in DB2 10 drive new value in resiliency through scalability improvements and fewer outages, whether those outages are planned or unplanned. Virtual storage enhancements deliver the ability to handle five to ten times more concurrent active users in a single DB2 subsystem than in previous releases of DB2 (as many as 20,000 concurrent active threads). Improved availability is supported by allowing more changes using schema evolution or data definition on demand. Security improvements also contribute to robust business resiliency.

Continuous availability enhancements

DB2 10 provides online schema enhancements that allow you to make changes to database objects (indexes and table spaces) while maximizing the availability of the altered objects. Through enhancements to ALTER statements, you can now change more attributes of indexes and table spaces without having to unload the data, drop and re-create the objects, regenerate all of the security authorizations, re-create the views, and reload the data. The changes are materialized when the altered objects are reorganized. DB2 10 allows fast changes of table space types, page sizes, data set sizes, and segment sizes. Conversion to universal table spaces is much simpler.

In addition, DB2 10 improves the usability and performance of online reorganization in several key ways. It supports the reorganization of disjoint partition ranges of a partitioned table space (also in DB2 9 now), and improves SWITCH phase performance and diagnostics. Also, DB2 10 removes restrictions related to online reorganization of base table spaces that use LOB columns.

Reduced catalog contention

In DB2 10, the DB2 catalog is restructured to reduce lock contention by removing all links in the catalog and directory. In addition, new functionality improves the lock avoidance techniques of DB2, and improves concurrency by holding acquired locks for less time and preventing writers from blocking the readers of data.

In DB2 10 new-function mode (NFM), you can access currently committed data to minimize transaction suspension. Now, a read transaction can access the currently committed and consistent image of rows that are incompatibly locked by write transactions without being blocked. Using this type of concurrency control can greatly reduce timeout situations between readers and writers who are accessing the same data row.

Virtual storage relief

Enhancements in DB2 10 substantially increase the capacity of a single DB2 subsystem by removing virtual storage and other constraints. This release moves most virtual storage above the 2 GB bar (64-bit addressing), which provides virtual storage relief and can greatly improve the vertical scalability of your DB2 subsystem while minimizing administration. In addition, a 64-bit ODBC driver is now available on DB2 9 and DB2 10.

Security enhancements

This release of DB2 provides critical enhancements to security and auditing, strengthening DB2 security in the z/OS environment. DB2 10 provides increased granularity for DB2 administrative authority. DB2 10 delivers a new DB2 data security that enables you to manage access to a table at the level of a row, a column, or both. In addition, you can define and create different audit policies to address the various security needs of your business.

1.2.3 Rapid application and warehouse deployment for business growth

SQL, IBM pureXML®, and optimization enhancements in DB2 10 help extend usability, improve performance, and ease application portability to DB2. DB2 10 delivers significant query improvements, with better performance and CPU reductions, allowing you to manage and maintain your data in a single platform infrastructure with single audit and security processes, and, most importantly, providing a single answer based on your core operational data.

SQL improvements

SQL enhancements deliver new function for improved productivity, DB2 family consistency, and simplify application porting to DB2 for z/OS from other platforms and database management systems. Enhancements are provided for SQL scalar functions and SQL table functions are added. Native SQL procedure language (SQL PL) is easier and faster. Implicit casting makes porting simpler, because DB2 SQL is more consistent with other products and platforms. Allowing more flexibility in the number of digits for fractions of seconds and allowing timestamps with time zones simplify porting. Moving sums and moving averages help in warehouse queries and in porting.

Temporal tables and versioning

In this release of DB2 for z/OS, you have a lot of flexibility in how you can query data based on periods of time. DB2 supports two types of periods: the system time (SYSTEM_TIME) period and the business time (BUSINESS_TIME) period. The SYSTEM_TIME period is a system-maintained period in which DB2 maintains the beginning and ending timestamp values for a row. For the BUSINESS_TIME period, you maintain the beginning and ending values for a row. Support of business time and system time allows for significant simplification of applications, pushing the complicated handling of these concepts down to the database engine itself.

In addition, DB2 10 introduces versioning, which is the process of keeping historical versions of rows for a temporal table that is defined with a SYSTEM_TIME period, or both time periods, allowing for simple retrieval of key historical data.

pureXML improvements

DB2 10 improves DB2 family consistency and productivity for pureXML users. These improvements also deliver excellent performance improvements. DB2 10 delivers binary XML format, XML schema validation as a built-in function, XML date and time data types and functions, XML parameters in routines, and much more.

1.2.4 Enhanced business analytics and mathematical functions with QMF

Query Management Facility™ (QMF™) Version 10.1 has new analytic and mathematical functions and OLAP support. It also provides access to any Java™ Database Connectivity (JDBC)-enabled database allowing a wider array of data (such as contents of IMS 11 Open Database) to be combined with DB2 within the same report.

1.3 Performance expectations for DB2 10

Customer measurements and laboratory benchmarks have shown a broad range of improvements in DB2 10. Figure 1-1, which assumes BIND and 1 MB page frames, shows the wide range of improvements found in IBM benchmark workloads. The OLTP workloads have substantial improvements for typical short running SQL transactions especially when RELEASE(DEALLOCATE) is used.

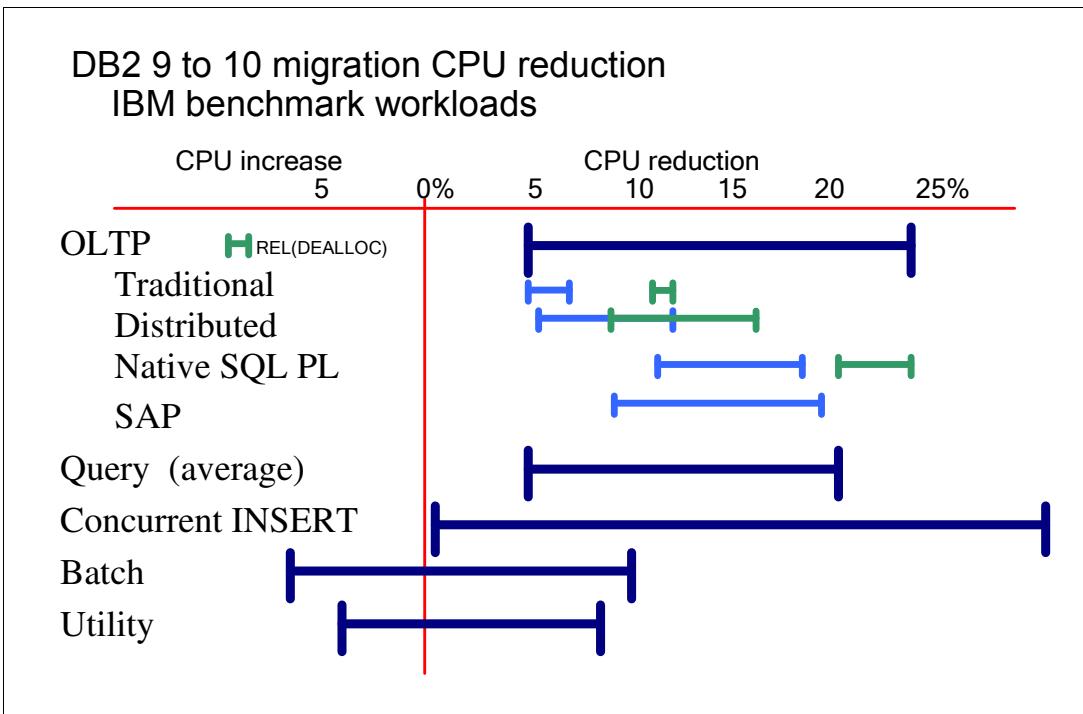


Figure 1-1 CPU reduction across some workloads

Transaction processing is usually part of the peak workload for customers. The range of CPU reductions was generally 5% to 25%. Traditional transactions running under IMS, after BIND, reduced CPU time by 5% to 7% initially, and can use RELEASE(DEALLOCATE) to reach 10% to 12%. Distributed transactions improved more ranging from 6% to 17%, with RELEASE(DEALLOCATE) as an important contributor. Native SQL procedure language that use common constructs such set statements and use SYSDUMMY1 improved by 10% to 20%, and 21% to 25% with RELEASE(DEALLOCATE).

SAP measurement results show that DB2 10 for z/OS delivers significant performance and scalability improvements compared to DB2 9 on both the IBM z10™ and z196 systems. SAP Sales and Distribution (SD) workload and SAP Day Posting workloads saw up to a 19% improvement in performance with DB2 10 compared to DB2 9. As the DB2 system got larger (more concurrent threads on more System z processors), there were larger DB2 10 performance improvements.

Using a small hardware configuration, there was up to 1.8 times improvement in performance with DB2 10 on a z196 compared to DB2 9 on a z10. Virtual storage in the DB2 DBM1 address space is no longer constrained. The maximum number of concurrent threads that can be connected to a single DB2 subsystem has increased by five to ten times with DB2 10. More extensive use of MAXKEEPD, an important performance parameter, is possible because of the virtual storage constraint relief. Significant reductions in latch contention were also observed across the board with both the SAP SD and SAP Day Posting workloads with DB2 10 compared to DB2 9. This reduction in latch contention allows DB2 10 to support the higher number of concurrent threads. It also helps boost the performance of DB2 10 as the DB2 system grows.

Queries saw a wide range of improvements, usually 5% to 20%. Concurrent insert work showed some of the best improvements, ranging from unchanged to more than 30%. The range for batch work spanned from a 7% increase to a 10% decrease, with most measurements showing a decrease, especially with inserts. Utilities also had some CPU

increases up to 5% and improvements of up to 8%, but most utility measurements were very close to those of DB2 9, improved by about 20% if migrating from V8.

The numbers in Figure 1-2 show some customer CPU measurements from the DB2 10 beta program. These measurements reflected various customer work better than a benchmark, but were usually less repeatable. Customers generally might not have dedicated resources, so they measured multiple times and checked for consistency of the runs. Most of the customer information showed the ability to get improvements similar to those in the benchmark measurements, with a wider range of work and results.

Sample improvements for CPU guesstimate

Run time CPU reductions	5% - 10%	
1 MB page size	0% - 5%	z10, z196
Page fix buffers	0% - 8%	V8 and high IO, in use?
Release deallocate	0% - 15%	short trans, batch
Virtual storage constraints	0% - 5%	memory, latches
Data sharing fewer members	1%	for each 2 members
Improved dynamic SQL cache	0% - 20%	literals
Insert	0% - 50%	high volume insert
Predicate evaluation	0% - 5%	complex predicates
Access: hash, index include	0% - 5%	access improved
Increased use of zIIP	0% - 3%	IO, RUNSTATS, parallel
Utilities (from V8)	3% - 20%	about the same as for 9 → 10
Productivity: memory, temporal, security, admin, ...		priceless

Figure 1-2 The CPU performance measurements from the beta program

The common range for CPU reductions is very wide. Understanding the magnitude of the gains for each individual customer and the breadth of applicability are important. It is also important to understand which gains are applicable to peak workload and therefore determining the charges for most pricing options.

When looking at a general workload, the performance expectation ranges from 5% to 10%. Transactions with only a few SQL statements save less, but can benefit from the increased ability to use RELEASE(DEALLOCATE). The change to use 1 MB hardware page frame sizes can be up to 5%, if you have a z10 or z196 and configure the LFAREA. 1 MB page frame size also depend upon page fixed buffers. Many customers have not yet taken advantage of the V8 page fix function, which can save up to 8% of the CPU time for I/O intensive workloads.

Virtual storage constraint relief is generally up to 5%, but extreme cases can save much more. Estimated saving is $\frac{1}{2}\%$ of CPU for each active member removed from data sharing.

Queries with many predicates can improve up to 60% but are more often in the range of 5% to 20%, depending upon whether the new optimizations and CPU reductions match the workload. Many customers reported insert improvements up to 40%, and larger compared to V8.

Increased use of zIIP comes from prefetch read, deferred write, most options of the RUNSTATS utility, and increased parallelism, and can give up to 3%.

Utilities in DB2 10 use roughly the same amount of CPU time as DB2 9, and much less than DB2 V8. The biggest benefits from DB2 10 are the productivity improvements in memory management, temporal SQL, security, and administration.

DB2 10 delivers by improving performance and reducing CPU usage. Most customers can achieve out-of-the-box CPU savings of 5% to 10% for traditional workloads and up to 20% for specific workloads described next. REBINDs are needed to obtain the best performance and memory improvements. DB2 reduces CPU usage by optimizing processor times and memory access, leveraging the latest processor improvements, larger amounts of memory, and z/OS enhancements. Improved scalability and constraint relief can add to the savings. Productivity improvements for database and systems administrators can drive even more savings.

In DB2 10, performance improvements focus on reducing CPU processing time without causing significant administration or application changes. Most performance improvements are implemented by simply migrating to DB2 10 and rebinding. You gain significant performance improvements from distributed data facility (DDF) optimization, buffer pool enhancements, parallelism enhancements, and more.

Early DB2 10 performance benchmarking and customer experience have shown a 5% to 10% CPU reduction in transactions after rebinding. Some customers might get more or some less CPU reduction depending on the workload. Customers who have scalability issues, such as virtual storage constraints or latching can see higher improvements. Opportunities for tuning can take advantage of memory improvements. More high volume, short-running distributed transactions can take advantage of CPU reductions, using RELEASE(DEALLOCATE).

Concurrent sequential insert CPU time can be reduced from 5% to 40%. Queries can be improved as much as 20% without access path change, and more for better access paths. A native SQL procedure workload has shown up to 20% CPU reduction using SET statements, IF statements, and SYSDUMMY1. Customers moving from DB2 9 can expect about the same CPU times for utilities, whereas customers moving from DB2 V8 will see CPU reductions up to 20%.

1.3.1 Insert performance

Insert performance was a highlight in both customer testing and benchmarking. The DB2 10 improvements built upon those of DB2 9, so that customers migrating directly from V8 get benefits from both. Some of the DB2 9 improvements required work by database administrators, such as larger index pages, the APPEND option, or use of the LASTUSED column for indexes. Others improvements are implemented in DB2: asymmetric index page split, log latch contention and LRSN spin reduction, and removal of the log write force.

DB2 10 improvements for these applications are automatic in CM, while some require database administrator work in NFM (index include columns and member cluster for universal table spaces).

Insert improvements vary, but some of the largest improvements in DB2 10 are for high volume concurrent sequential inserts in a data sharing environment.

A specific test case shows the performance of sequential key inserts into 3 universal table spaces with range partitioning. Many (240) JDBC clients are inserting concurrently in 2 way data sharing, using multi row insert with member clustering. The CPU for this case drops by about a factor of 9. The insert rate improves by about a factor of 6.

DB2 10 improves insert performance in many ways:

- ▶ Space search improvement: DB2 10 changes the behavior of space management algorithms when inserting a row to benefit sequential inserts.
- ▶ I/O parallelism for index updates: DB2 10 can read leaf pages in parallel when a table is defined with three or more indexes.

- ▶ Log latch reduction.
- ▶ Faster log I/O reduces commit suspension time.
- ▶ Log record sequence number spin avoidance for inserts to the same page: In DB2 10 NFM, consecutive log records for inserts to the same data page can have the same LRSN value.
- ▶ Referential integrity checking improvement: DB2 10 changes, when inserting a row into a dependent table. in some cases it is not necessary to access the parent key for referential constraint checking.
- ▶ Member clustering is permitted for universal table spaces: Database administrators can ALTER the table space type for table spaces containing a single table space to become universal table spaces and to use member clustering.

With all of these changes, inserts improve substantially in CPU and suspension time, and performance is more uniform across a range of table space types. The largest improvements are for sequential inserts with contention, but random inserts also see rather large improvements when contention is present.

Universal table space insert performance gets very close to that of classic partitioned and segmented table spaces, except for universal table space sequential insert with row level locking. For high performance inserts that experience contention in data sharing with universal table spaces, you can use the member cluster option.

1.3.2 When is it necessary to REBIND

We discuss a scenario for the IRWW benchmark transaction that is run on DB2 9, then on DB2 10. This scenario uses some new functions in DB2 9 to BIND or REBIND a package with access control management to allow three copies. These are fairly light IMS transactions that have been used for many DB2 transaction benchmarks. See Figure 1-3.

Measurements of IBM Relational Warehouse Workload (IRWW) with data sharing

- Base: DB2 9 NFM REBIND with PLANMGMT EXTENDED
- DB2 9 NFM → DB2 10 CM without REBIND showed 1.3% CPU reduction
 - DB2 10 CM REBIND with same access path showed 4.8% CPU reduction
 - DB2 10 NFM brought 5.1% CPU reduction
 - DB2 10 CM or NFM with RELEASE DEALLOCATE 12.6% CPU reduction from DB2 9

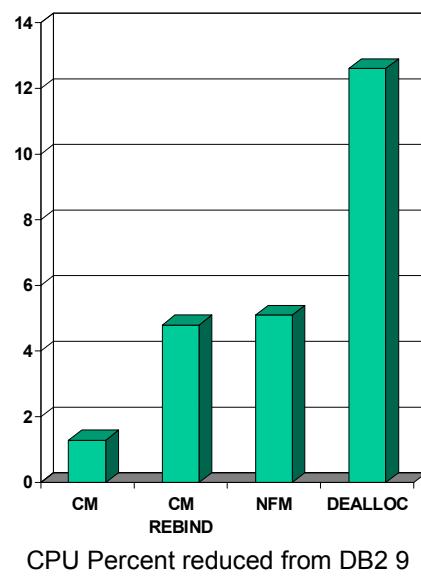


Figure 1-3 Impact of binding on IRWW workload

In step 1, this application is moved to DB2 10 CM without a REBIND, and the result is a 1.3% reduction in CPU time.

In step 2, still in DB2 10 CM, a REBIND is performed but with exactly the same access path. With the REBIND, the CPU savings over DB2 9 is 4.8%.

In step 3, moving to NFM, the CPU time is about the same.

In step 4, these transactions are changed to use RELEASE(DEALLOCATE), saving 12.6% of the total CPU time compared to DB2 9.

This scenario demonstrates the runtime improvements and CPU value of REBIND and RELEASE(DEALLOCATE) for a high volume, short transaction. Using RELEASE(DEALLOCATE) provides the opportunity for significant performance improvement.

REBIND is not required for migration to DB2 10, but REBIND is highly desirable. Getting the best performance improvements and eliminating regression does depend upon rebind in most situations: getting current structures, better access paths, and reusing threads. Eliminating performance regression might depend upon REBIND. Storage constraint relief depends upon REBIND. Changing to use RELEASE(DEALLOCATE) requires a REBIND.

All plans containing DBRMs must be rebound. All packages that were last bound on V5 or lower must be rebound. Static SQL statements with DEGREE(ANY) for parallel processing need to be rebound, or it will be serial. Other REBINDs can be staged over weeks of time, and REBIND is only needed once per package for the migration. Improvements in access paths can be very significant, such as stage 2 predicates that can become stage 1. REBIND in DB2 10 takes more CPU and elapsed time than in prior versions, but more concurrent REBINDs are possible in NFM. So, in conclusion: REBIND.

1.3.3 What else is needed to get performance out-of-the-box

Some customers are getting a lot of performance improvements from CM on the first day. The improvements do require a REBIND in most situations, and that does mean checking and testing, but DB2 version changes also take testing, so combining the work for a dramatic improvement will work for many customers. This change can be implemented for a few very high volume transactions and provide a great return. With the IRWW workload, using static SQL, the improvement from use of RELEASE(DEALLOCATE) is about 12% as compared to DB2 9 using RELEASE(COMMIT).

RELEASE(DEALLOCATE) cannot be used by distributed applications before DB2 10. Improvements have been implemented to recycle threads every 200 transactions. RELEASE(DEALLOCATE) is best for high volume batch or transactions with few SQL statements in each COMMIT. The transactions also need to be well-behaved for locking. Local transactions need some mechanism to end the thread, so that utilities, data definition changes, and other processes can be performed.

RELEASE(DEALLOCATE) has been part of DB2 for a long time, but DB2 10 makes the function more useful. Using RELEASE(DEALLOCATE) requires much more memory. The dramatic memory improvements in DB2 10 that you can achieve with rebind makes it possible to use RELEASE(DEALLOCATE) more. This change saves significant CPU time for high volume transactions with few short running SQL statements, without changing applications or DDL. RELEASE(DEALLOCATE) works in CM, but does require a REBIND unless the packages already use RELEASE(DEALLOCATE). For DDF work, after rebinding with RELEASE(DEALLOCATE), the customer must issue the MODIFY DDF PKGREL(BINDOPT) command. By default JDBC is changed to use RELEASE(DEALLOCATE).

The High Performance DBAT function will cause resources to be periodically released after 200 commits. So RELEASE(DEALLOCATE) is more applicable and safer to use in many more situations. RELEASE(DEALLOCATE) depends upon having very well debugged, well behaved applications that are careful with locking and commit frequently.

1.3.4 DB2 10 improvements for RELEASE(DEALLOCATE)

RELEASE(DEALLOCATE) takes more virtual storage, which is not available for many customers on DB2 9 or V8. In DB2 9, you get RELEASE(COMMIT) for DDF work, even if packages are bound RELEASE(DEALLOCATE). DB2 10 allows distributed RELEASE(DEALLOCATE). DB2 10 also changes to end the thread every 200 transactions. If your distributed transactions are already bound with RELEASE (DEALLOCATE), then they do not need to be rebound, just change the DDF setting PKGREL=BNDOPT by using the new MODIFY DDF command (see 8.1, “High performance DBATs” on page 240.)

If your application is not well behaved, then you can get into problems, deadlocks, timeouts, and inability to run utilities. If your process is missing commits or takes gross table space locks, then it is not a good candidate for DEALLOCATE.

Some customers found longer BIND times in DB2 10. The default for access path management has changed in DB2 10 from none to EXTENDED. Customers who move from DB2 9 and have used access path management have some improvements. Customers who did not have access path management will find increases in BIND CPU time from this change. If you want to reduce the time for BIND, then change the subsystem parameter back to PLANMGMT(OFF). Use access path management where you have noted problems with new access paths.

1.3.5 Performance estimation

If you need a rough estimate of performance expected, then you can compare your workloads to those in this book. If you want an accurate estimate of performance for your workloads, then you can benchmark your workload and measure the performance. Variations in workloads, software, hardware, options and tuning have been substantial, as noted in presentations and papers.

While we see most customers getting 5% to 10% improvements, some get more and some get less. Some customer results are noted in presentations, and more will be shown in upcoming conferences. Your workloads, your hardware, your software, your options, and your tuning might differ from these measurements. Although we understand the desire to have a simple, reliable, accurate estimate of performance, the combination is not possible.

1.4 How to read this book

First of all, let us establish what this book is not, and what it is.

It is not a general DB2 performance and monitoring manual; there are other standard manuals and standard education classes to provide this. Rather, this book assumes that the reader already owns skills on DB2 for z/OS performance concepts, monitoring, and tuning.

It is not a general DB2 10 functional description manual. The topic of a new version of DB2 is vast and DB2 10 has something new for everybody. See *DB2 for z/OS Technical Overview*, SG24-7892 for an overview and the bibliography mentioned in that book. Or attend a DB2 10 for z/OS Transition Class.

Now let us see what this book is. It is a technical report of detailed performance measurements of several performance related topics of DB2 10 for z/OS, often compared to DB2 9.

The measurements are meant to show how the topics perform, and the topics are not just specific to performance enhancement. They can be about new functionalities, and in this case they can be compared to similar functions or to the behavior without the function.

Hopefully you will get an understanding of the function and a realistic expectation of what the performance will be.

All measurements were done in a laboratory environment, and as such, they might be atypical for being run in a dedicated environment and focused on a specific function, with no other workload causing resource contention. Extrapolation and generalization require judgement.

Another aspect to clarify is the reasoning behind the structure of the table of contents. We tried to follow the general technical area of the products as much as possible but often ended up with a chapter being too large or a critical topic being spread over too many chapters. So we kept tweaking the contents over the years and tried our best to avoid duplication by providing cross references. Here is a quick road map to the book structure:

- ▶ Chapter 1. Introduction:

This chapter presents the executive summary, including “all you wanted to know about DB2 10 performance” in very few pages.

- ▶ Chapter 2. Subsystem:

Most of the many new DB2 engine functions are described here. Several do not require human intervention.

- ▶ Chapter 3. Synergy with z platform:

This area might be considered together with the Subsystem chapter, but it grew too large, and it is certainly important to stress the synergy with the z platform in a separate chapter.

- ▶ Chapter 4. Table space design options:

There were so many functions related to table space structures in DB2 10—universal table space, LOBs, XML, hash table—that we decided to dedicate a chapter to these topics. Here you will find lots of ideas on how to apply and make use of these new DBA tools.

- ▶ Chapter 5. Sample workloads:

We gathered measurements on known workloads to put a few stakes in the ground to set the expectations for when you migrate. We added considerations on virtual storage and intensive insert processing. Traditionally, the explosion of intensive insert workloads has been critical for many users, highlighting issues and bottlenecks, most of which have been removed by several DB2 functional enhancements.

- ▶ Chapter 6. SQL:

Here are described several enhancements in the access path selection, transparent to the application developer. We add implicit casting between strings and numerics, and some enhancements in SQL for native SQL procedures.

- ▶ Chapter 7. Application environment:

This chapter is all about new application functions, the use of which will substantially improve the productivity of application developers.

- ▶ Chapter 8. Distributed environment:

This chapter includes functions that in other times were called e-business.

- ▶ Chapter 9. Utilities:
Utilities have improved in functions and integration with disk storage (zSynergy again). The use of IBM FlashCopy® at the object level is supported for Copy and Recover.
- ▶ Chapter 10. Security:
DB2 10 has added several new security functions. Here we describe the performance impact of utilizing the new policy-based audit capability and the support for row and column access control.
- ▶ Chapter 11. Installation and migration:
This chapter provides measurements of migration and skip-migration steps for sample DB2 catalogs.
- ▶ Chapter 12. Monitoring and Extended Insight:
Here we show how DB2 10 has enhanced instrumentation and, with the new Extended Insight option of IBM OMEGAMON® PE¹ we can provide end to end monitoring support.
- ▶ Appendix A. Recent maintenance:
This appendix is an attempt at keeping up with what the maintenance stream is adding in terms of functions and performance after the general availability of DB2 10.

¹ In this section (and throughout the book) we use OMEGAMON PE as a reference to the product IBM Tivoli® OMEGAMON XE for DB2 Performance Expert Version V5R1.



Subsystem

DB2 10 continues to evolve, removing structural constraints in order to support its increasing use by concurrent workloads. Several improvements to the DB2 engine allow faster accesses for applications and data base administration.

In this chapter, we discuss the following topics:

- ▶ Catalog restructure
- ▶ Latching contention relief
- ▶ Dynamic prefetch enhancements
- ▶ Buffer pool enhancements
- ▶ Work file enhancements
- ▶ Logging enhancements
- ▶ I/O parallelism for index updates
- ▶ Space search improvement
- ▶ Log record sequence number spin avoidance for inserts to the same page
- ▶ Compression on insert

You can find the functional details of these topics in the IBM documentation and in the Redbooks publication *DB2 10 Technical Overview*, SG24-7893.

2.1 Catalog restructure

In DB2 10, the catalog and directory table and related index spaces are SMS managed. The main reason for this change is that some of the DB2 catalog table spaces are partitioned by growth with a DSSIZE greater than 4 GB. DSSIZE requires DFSMS Extended Addressability (EA), and EA requires the data set to be system managed.

With the DB2 catalog and directory data set SMS managed, DB2 automatically exploits SMS features that are only available to SMS managed data sets. Data set attributes, performance characteristics, and management rules of data sets are transparently defined in SMS using data classes (DATACLAS), storage classes (STORCLAS), and management classes (MGMTCLAS). The assignment of DATACLAS, STORCLAS, and MGMTCLAS can be externally provided (for example, in the IDCAMS DEFINE CLUSTER command) and enforced through SMS policies, also known as automatic class selection (ACS) routines:

- ▶ **STORGRP:**

DASD volumes are grouped into SMS storage groups (STORGRP). During data set creation, the STORGRP assignment is transparently enforced by the SMS policy through the STORGRP ACS routine. Within a selected STORGRP, SMS places the data set that is to be created onto one of the volumes within that SMS STORGRP. To prevent an SMS STORGRP from becoming full, you can provide overflow STORGRPs, and you can define utilization thresholds that are used by SMS to send alert messages to the console in case the STORGRP utilization threshold is exceeded. Monitoring overflow STORGRPs and automating SMS STORGRP utilization messages provide strong interfaces for ensuring DASD capacity availability. You can also use the information provided by IDCAMS DCOLLECT for regular STORGRP monitoring and capacity planning.

DFSMS assigns all of the volumes in the STORGRP to either a primary candidate list or a secondary candidate list. Many customers prefer that DFSMS randomly pick a volume among all of the volumes that have sufficient space for the new data set. DFSMS will not do this if there are such volumes on the primary candidate list, but it will do this if the primary candidate list is empty. One way to cause the primary candidate list to be empty is to specify some value for Initial Access Response Time (any value will do, as long as it is non-blank.)

Spreading the DB2 catalog and directory across the entire STORGRP is not important. If you want to know exactly where the catalog data sets are, you can dedicate a STORGRP to the DB2 catalog and directory, starting with a small size, with just enough volumes to support all of the DB2 systems. You can later add volumes to the STORGRP as the catalog grows or as there are more DB2 systems.

Another potential use of STORCLAS is the SUSTAINED DATA RATE, useful if you want *user* data sets striped.

- ▶ **DATACLAS:**

Data set attributes are transparently assigned through SMS data classes (DATACLAS) during data set creation. The data class is where you have to specify the Extended Addressability attribute required for some of the DB2 catalog data sets. New SMS data set attributes that are required to support better I/O performance and data set availability can transparently be activated by changing online the DATACLAS in SMS. For example, you can change online the DATACLAS volume count which becomes active immediately for all data sets using that DATACLAS. There is no need for you to run an IDCAMS ALTER command to add volumes to the data set as you might have to if the data sets were non-SMS managed.

► STORCLAS:

The use of SMS storage classes (STORCLAS) allows you to assign performance attributes at data set level, because an SMS STORCLAS is assigned during data set creation. With non-SMS managed data sets, storage related performance attributes are normally only available on a volume level affecting all data sets residing on that volume. For example, you can support a minimum disk response time for a particular data set by assigning a particular SMS STORCLAS that supports that performance requirement.

Some data set attributes (for example, the volume count) can simply be adjusted by changing the SMS data class. For example, instead of using the ALTER command to add volumes to an existing DB2 VSAM LDS data set, you can simply increase the volume count in the data class definition that is used by the DB2 VSAM LDS data set.

SMS storage can very effectively be used for the DB2 catalog and directory if all of the default parameters are used for the SMS constructs, except for Extended Addressability. Ordinarily only STORCLAS is required, but in this case, DATACLAS is required too. MGMTCLAS is not required, but it is useful if DFSMShsm is going to manage these volumes.

2.1.1 Catalog changes

Figure 2-1 shows how the DB2 catalog continues to grow from release to release.

DB2 Version	Table spaces	Tables	Indexes	LOBs	Columns	Table check constraints
V1	11	25	27	0	269	N/A
V3	11	43	44	0	584	N/A
V5	12	54	62	0	731	46
V6	15	65	93	0	967	59
V7	20	84	118	2	1212	105
V8	22	85	132	2	1265	105
DB2 9	28	104	165	3	1643	119
DB2 10	95 (104-9)	134	233	18	1922	119

Figure 2-1 DB2 catalog evolution

DB2 10 reduces catalog contention by eliminating catalog links and converting the catalog tables to use row-level locking instead of page-level locking. These changes, made available in new-function mode, reduce catalog contention dramatically during such events as concurrent DDL processing and BIND operations. Although BIND and REBIND might take longer to process, they cause only minimal contention.

DB2 catalog table spaces also change from many tables to one table per table space, which also reduces table space contention, in a partition by growth table space defined as DSSIZE 64 GB and MAXPART 1.

Rather than repeating columns with parts of long strings, the catalog uses CLOB and BLOB columns to store the data, expanding maximum sizes. Inline LOBs are used for the performance improvements. The new structure allows more standard processes, so that all catalog tables can be reorganized and checked online.

The Redbooks publication, *DB2 10 for z/OS Technical Overview*, SG24-7892 discusses the items related to the restructuring of the DB2 catalog in DB2 10.

Important: If your applications do not use LOBs today and if you are not accustomed to LOBs, be aware that DB2 10 catalog now has many more LOB objects. The restructuring and the presence of LOBs imply changes to catalog recovery procedures and new disaster recovery testing. This is specially important for point in time recovery of the DB2 catalog and directory.

For migrated systems, the catalog and directory restructuring happens during the DSNTIJEN job. For new installations, the catalog and directory is NFM ready.

The structure of the DSNTIJEN job is similar to the one provided in previous versions of DB2. It is started with CATENFM START to enter ENFM, followed by CATENFM CONVERT and REORG for every catalog and directory table space. The DSNTIJEN job can be halted at any time during execution, and there is no need to modify the job before resubmitting it. It automatically skips the steps that were previously completed. A special REORG is run in the DSNTIJEN job, and the 'CONVERTV10' keyword is used to indicate it is the special ENFM REORG.

For more information about the migration process, see 11.3, "Migration" on page 305.

2.1.2 Impact of DB2 catalog migration

Migrating a DB2 catalog to DB2 10 conversion mode (CM9 or CM8) includes the creation of indexes that replace catalog links. Indexes are used by DB2 for accessing the DB2 catalog, but the catalogs links are kept and maintained in case they are needed for a version fallback.

One of the DB2 catalog related benefits of migrating to DB2 10 NFM is an increase in concurrency for DDL and BIND operations. Row level locking is used for the DB2 catalog tables in NFM.

The utilization of indexes for the catalog instead of links improves concurrency and makes catalog health checking easier. The use of DSN1CHKR is no longer required. As a consequence of this change, however, operations that access the DB2 catalog are expected to show a higher CPU utilization.

To observe individual PREPARE performance regression, we measured the execution of a mixed dynamic SQL query workload. We used the time of IFCID 64 to IFCID 58 to collect the PREPARE elapsed and CPU time. IFCID 64 is a performance record. This IFCID records the start of the execution of a PREPARE SQL statement. When the execution of this statement ends, IFCID 58 is written. This record is written at the application server only when the DRDA protocol is used. This record is written when performance trace class 3 is on. You can check the member *hlq.SDSNIVPD(DSNWMSGS)* for DB2 10 IFCID field descriptions.

The dynamic SQL full PREPARE increase in class 2 CPU and elapsed time ranges from 20% to 30% when comparing DB2 9 to DB2 10. This extra cost is largely attributed to the utilization of indexes instead of links in the DB2 catalog. Increased complexity to access path selection contributes to some degree to the observed increase. The incremental cost varies depending on the statement type and complexity; the increased CPU time is more evident for short running queries.

Important: Catalog indexes are introduced when you migrate to CM but there is no increased catalog concurrency in this mode. Because you pay the price of having them but you do not get the benefits, consider moving to NFM fast if catalog contention is a concern in your organization.

2.1.3 DDL performance and concurrency

We used a workload consisting of mixed CREATE statements of table spaces, tables, and indexes in a single database for testing DDL performance.

We observed that a single thread DDL process in DB2 10 CM9 shows equivalent performance to DB2 9. The same process in DB2 10 NFM exhibits a reduction in elapsed time on the order of 4% and an increase between 10% and 40% in CLASS 2 CPU time.

A DROP database command involving hundreds of objects has equivalent elapsed time but an increase in CLASS 2 CPU in the range of 10% to 30% when comparing DB2 10 CM9 to DB2 9. In NFM, the same DROP statement shows a reduction in elapsed time of about 13% to 20%, and an increase in CLASS 2 CPU time of 63% to 75%, about 10 times the number of GETPAGES, and approximately 2 times the number of LOCKS.

In general DDL is not considered a performance sensitive function but these tests show the impact of the new DB2 catalog structure in DDL operations. While elapsed time is reduced in DB2 10 NFM compared to DB2 9, CLASS 2 CPU times increases significantly, especially when hundreds of objects are involved. The increase in CLASS 2 CPU time is attributed to the existence and maintenance of new catalog indexes.

DB2 10 NFM provides improved concurrency for parallel DDL. A series of tests executed on a System z z10 LPAR with 5 CPUs consisting of 5 parallel DDL processes showed that the concurrency enhancements in DB2 10 provides almost linear scalability. During this test each stream created a set of 100 objects (explicit table space, a table and two indexes) with an explicit COMMIT after each set of objects (table space, table and indexes). In DB2 10 NFM, 5 parallel streams, each one with 100 objects, uses 30% of the elapsed time required by a single process creating 500 objects with DB2 9.

While simple DDL streams with frequent COMMITS were executed concurrently with success, contention problems can still appear in DB2 10 NFM: complex streams that include the implicit creation of table spaces, such as for LOB and XML columns, might still generate contention that is not solved in DB2 10. One example is the occurrence of deadlocks on SYSTSTAB for concurrent DDL streams creating objects with LOB and XML columns.

In order to minimize the risk of concurrency issues in DB2 10, you still need to consider the following guidelines:

- ▶ Frequent COMMITS are necessary, especially for complex DDL streams.
- ▶ CREATE and DROP database can collide on a DBD lock and expose a TIMEOUT situation. Consider serializing these operations.
- ▶ CREATE of tables that result in the implicit creation of tables spaces such LOBs and XML columns can get deadlocks. Consider serialization for aggressive process involving these operations.

2.1.4 BIND and REBIND stability and performance

DB2 10 builds on the access path stability improvements that were offered in DB2 9, introducing a more comprehensive framework for the management of access paths.

With these new features for access path stability, you can capture information for storage in an access path repository, save multiple copies of access paths, and switch between different copies of access paths of the same query. For both bind and rebind processes, you can regenerate runtime structures without changing access paths. In addition, you can compare the new and old access paths at bind or rebind processing and indicate that a warning or error is issued when an access path changes.

See *DB2 10 for z/OS Technical Overview*, SG24-7892 for details on this feature.

DB2 10 access plan stability support addresses the following issues:

- ▶ Prior to DB2 10, the SYSPACKAGE catalog table contains only the information about the current version. No information is available for previous or original copies. Users must use SWITCH(PREVIOUS) or SWITCH(ORIGINAL) to reveal the SYSPACKAGE data for that version.

DB2 10 adds the new table SYSPACKCOPY to the catalog. This table keeps package information about previous and original packages identified by column COPYID (1 - previous copy, 2 - original copy).

- ▶ Prior to DB2 10, when redundant package copies are saved (that is, the cases where the access path does not change), SPT01 space is wasted.

DB2 10 for z/OS introduces the APRETAINDUP REBIND option that determines whether or not DB2 retains an old package when access paths of the old copy are identical to the incoming (that is, the newer) copy.

This option applies only when PLANMGMT(BASIC) or PLANMGMT(EXTENDED) is in effect. APRETAINDUP(YES) is the default, which causes DB2 to always retain older package copies. This use is compatible with DB2 9. However, if APRETAINDUP(NO) is used, DB2 only retains the newer copy and this provides savings in disk space.

- ▶ DB2 10 supports the following REBIND PACKAGE options for the *native SQL stored procedure* packages:
 - PLANMGMT
 - SWITCH
- ▶ PLANMGMT=OFF is the default value in DB2 9. In DB2 10, the default is changed to PLANMGMT=EXTENDED.

Important: DB2 10 changes the default for PLANMGMT from OFF (in DB2 9) to EXTENDED. You can observe BIND and REBIND performance regression in DB2 10 if you were using the defaults in DB2 9. You can set PLANMGMT=OFF in DB2 10 if you do not need the benefits provided by the DB2 10 defaults.

When migrating from DB2 V8 to DB2 9 or DB2 10 with environments without disk space constraints, use PLANMGMT(EXTENDED) for the first REBIND on migration to DB2 9. Also use PLANMGMT(EXTENDED) for any subsequent REBINDs on DB2 9. This mechanism preserves V8 packages as ORIGINAL and older DB2 9 packages as PREVIOUS.

DB2 10 also introduces a new DSNZPARM PLANMGMTSCOPE (defined together with PLANMGMT in install panel DSNTIP8). Out of the possible values, ALL, STATIC, and DYNAMIC, currently the only permitted value is STATIC.

BIND and REBIND performance: Single thread

Because PLANMGMT=OFF is the default in DB2 9 and PLANMGMT=EXTENDED is the default in DB2 10, REBIND might show performance regression.

Measurements were performed in a single thread non-data sharing environment to evaluate the difference between REBIND on DB2 9 and DB2 10.

The measurements show the following results:

- ▶ DB2 9 PLANMGMT(OFF) versus DB2 10 CM9 and NFM PLANMGMT(EXTENDED)
Elapsed time increase from 100% to 200%. CLASS 2 CPU time increase from 50% to 70%.

- ▶ DB2 9 PLANMGMT(OFF) versus DB2 10 CM9 PLANMGMT(OFF)
Both elapsed time and CLASS 2 CPU time increase from 20% to 40%.
- ▶ DB2 10 CM9 PLANMGMT(OFF) versus DB2 10 NFM PLANMGMT(OFF)
Elapsed time increase of 11%. CLASS 2 CPU time increase of 5%.

Individual BIND performance is very close to REBIND performance.

Performance of concurrent BIND and REBIND in NFM

Using a z10 System z with 5 CPUs, we explored the performance of concurrent BIND and REBIND and NFM.

Our test scenario consisted of a non-data sharing environment running separate sets of REBINDs of different packages processes. Each one of them had more than 200 packages in 4 separate threads. The runs were allowed to execute concurrently with no timeout or deadlock reported. While concurrency is allowed, scalability is not linear: you need 2 to 3 parallel streams to do the same amount of REBIND throughput in DB2 10 NFM compared to DB2 9.

In a data sharing environment, group buffer pool dependency can further slow concurrent rebinds. APAR PM24721 (PTF UK63457) can improve catalog and directory LOB insert performance in data sharing significantly.

With PM24721 the space search algorithm has been changed to cache the lowest space map page from the last inserted record if the object is DBD01 or DBD06. Any subsequent insert will search the data from the cached space map page until the prior deleted space is committed. This process can repeatedly avoid searching the uncommitted delete space created by the BIND/REBIND process and further reduce getpage activity and CPU time, and is less likely to append the new row at the end of the table, hereby reducing the space growth. When the deleted space is committed, the space search algorithm resumes its original process for any subsequent insert. Thereafter, the same process is repeated and a new cached space map page will be established. The logic has been changed to update the NPAGE value of RTS in-memory control block during operation of LOB data.

Without this APAR, repeated REBINDs with GBP dependency can exhibit 2 to 3 times more elapsed and CPU time.

2.1.5 Compression and inline LOBs for SPT01

In some scenarios, DB2 9 users have come close to reaching the 64 GB size limit for SPT01. DB2 9 supports SPT01 compression by application of the APAR PK80375. This APAR added an opaque DB2 subsystem parameter called COMPRESS_SPT01 to DSN6SPRM that can be used to indicate whether the SPT01 directory space needs to be compressed.

DB2 10 restructures SPT01 to allow storing packages in LOBs. SPT01 is split into several pieces with the larger sections of each package stored in two LOBs. This greatly expands the number of packages that can be stored in SPT01. However, it makes compression ineffective on packages because LOBs cannot be compressed. DB2 10 also moves SPT01 to the 32 KB buffer pool to further position to use inline LOBs.

A DB2 9 catalog with compressed SPT01 will expand after migration to DB2 10 CM and require more DASD storage. In some cases, we observed a 64 GB compressed SPT01 expanding to more than 80 page sets in the new LOB table space SYSSPUXA where there is no compression available.

APAR PM27073 (UK65379) optionally enables the use of inline LOBs for SPT01. The use of inline LOBs for SPT01 helps reduce overall space requirements and can improve BIND performance, but on the other hand, using inline LOBs makes it more likely that the 64 GB limit will be reached. Inline LOBs must not be used for SPT01 if the 64 GB limit is an issue.

APAR PM27073, sets the bases for the implementation of inline LOBs in SPT01. PM27073 is the pre-conditioning APAR for support of inline length of DSNDB01.SPT01 which is enabled by APAR PM27811. At this time, the only change is new output in the DISPLAY GROUP DETAIL message DSN7100I. In a data sharing group, this pre-conditioning APAR must be applied to all members before the enabling APAR PM27811 is applied to any member. The enabling APAR adds the ability to change the inline length of DSNDB01.SPT01.

This APAR adds a new DB2 subsystem parameter in DSN6SPRM called SPT01_INLINE_LENGTH to specify the maximum length in single- byte characters of LOB column data in the SPT01 directory space to be maintained in the base table. SPT01_INLINE_LENGTH is externalized on installation panel DSNTIPA2 as SPT01 INLINE LENGTH.

The new DSNSZPARM is SPT01_INLINE_LENGTH and the maximum value is 32138. The valid settings are an integer from 1 - 32138 or NOINLINE, where NOINLINE means that no LOB data is to be placed inline in the SPT01 base table. The default setting is 32138.

When DB2 is started for the first time with APAR PM27811 applied, the in-line length of DSNDB01.SPT01 is changed from 0 to 32138 and the table space DSNDB01.SPT01 is put in Advisory REORG-pending (AREOR) status. You need to REORG SPT01 with SHRLEVEL(REFERENCE) to materialize the inline LOB data.

- ▶ If you decrease the setting, DB2 will place SPT01 in REORG- pending (REORP) status after you bring the change on-line.
- ▶ When the in-line length is made larger, a new message DSNG010I will be issued showing the new length and the table space will be put in AREOR. New rows inserted by BIND will use the new length and a REORG will convert all the rows to the new length.

A change to the SPT01_INLINE_LENGTH parameter does not take effect until you use the -SET SYSPARM command to bring it online. In other words, even if you start or restart DB2 after changing the value, the change is not honored until you issue the -SET SYSPARM command.

When changing the setting of SPT01_INLINE_LENGTH for a data sharing group, make the same change on all members before running the -SET SYSPARM command on any member.

The APAR also externalizes the existing parameter in DSN6SPRM, COMPRESS_SPT01 on the DB2 installation panel DSNTIPA2 as COMPRESS_SPT01. The COMPRESS_SPT01 parameter indicates if DB2 is to compress data in the SPT01 base table. Valid settings are NO and YES. The default is NO.

2.2 Latching contention relief

The latch is a DB2 mechanism for controlling concurrent events or the use of system resources. Latches are conceptually similar to locks in that they control serialization. They can improve concurrency because they are usually held for a shorter duration than locks and they cannot *deadlatch*. However, latches can wait, and this wait time is reported in accounting trace class 3 data.

Most DB2 latches that can impact scalability have an improvement with DB2 10 CM:

- ▶ LC12: Global Transaction ID serialization
- ▶ LC14: Buffer Manager serialization
- ▶ LC19: Log write in both data sharing and non data sharing (CM and also NFM)
- ▶ LC24: EDM thread storage serialization (Latch 24)
- ▶ LC24: Buffer Manager serialization (Latch 56)
- ▶ LC27: WLM serialization latch for stored procedures and UDF
- ▶ LC32: Storage Manager serialization
- ▶ IRLM: IRLM hash contention
- ▶ CML: z/OS Cross Memory Local suspend lock
- ▶ UTSERIAL: Utility serialization lock for SYSLGRNG (removed in NFM)

In this section we discuss the performance implications for the following contention relief:

- ▶ Latch class 19
- ▶ Latch class 24 (EDM)
- ▶ Latch class 32
- ▶ UTSERIAL elimination

2.2.1 Latch class 19

Latch class 19 contention can limit workload scalability in environments where there is a large number of log records created in the same interval.

Since DB2 Version 1, a single latch was used for the entire DB2 subsystem to serialize updates to the log buffers when a request is received to create a log record. The basic process is as follows:

1. The latch is obtained.
2. The RBA range is allocated.
3. The log record is moved into the log buffer.
4. The latch is released.

This method simplifies processing but also creates a bottleneck when CPUs get faster and the number of CPUs on a system increases. Recent tests in IBM labs were able to hit several hundred thousand log latch contentions per second on a 5-way z10 system with DB2 9.

With DB2 10, we still have the log latch, but it is held for the minimum time necessary to allocate space for the log record. The movement of the log record and updating of the control structures is done after the latch is released, allowing multiple log records to be moved in parallel, and improve throughput.

2.2.2 Latch class 24 (EDM)

Each EDM pool has a single latch to control changes for getting, freeing, or LRU shifting of EDM storage objects. This single latch is particularly a huge bottleneck in previous releases where there is frequent package section allocation for RELEASE(COMMIT) applications that typically have very short execution times and have frequent COMMITs. When many threads are simultaneously trying to allocate a package section, they frequently are suspended on this single latch waiting for another thread to complete using the same latch.

Because EDM thread storage cannot be shared among multiple threads, there is really no reason to allocate it in shared storage (EDM pool), which requires latching to serialize updates to mapping control blocks. So in DB2 10, the EDM thread storage pools (one below the bar and one above the bar) are eliminated. Instead EDM allocates plan and package structures from the threads agent local pools. Because agent local pools are not shared, no latch at all is needed, and this bottleneck is eliminated.

The DSNZPARM EDMPOOL for the below the bar pool remains as an option. Instead of allocating this storage at DB2 startup, it now acts as a limit on allocating. There have been problems in the past where some loop or runaway situation was able to fill up the EDM pool. So by having this option to limit the storage, the DB2 subsystem can survive by not having all below the bar storage over-allocated.

For comparison to previous releases, the CT and PT page counts converted to bytes of storage are added in the EDM QISE statistics block in IFCID 2. OMEGAMON® PE shows this as a value in below the bar and again above the bar on STATISTICS REPORT like the following example:

THREAD PLAN AND PACKAGE STORAGE	(MB)
---------------------------------	------

THREAD PLAN AND PACKAGE STORAGE	(MB)
---------------------------------	------

The storage below the bar associated with CT and PT pages only occurs for plans and packages that are not rebound on DB2 10. When they are rebound, this storage is allocated above the bar. One exception is the column procedure code specific to each statement that is allocated below the bar but only during its use.

Figure 2-2 demonstrates a situation with high LC contention in DB2 9 and how the elimination of the EDM thread storage pools improves performance. The application references 100 packages and COMMITS after executing them all. Each transaction does this 100 times for a total of 10k package allocations. The test is run with 100 concurrent threads doing this activity. The measure is how many can complete in the 3 minute time limit.

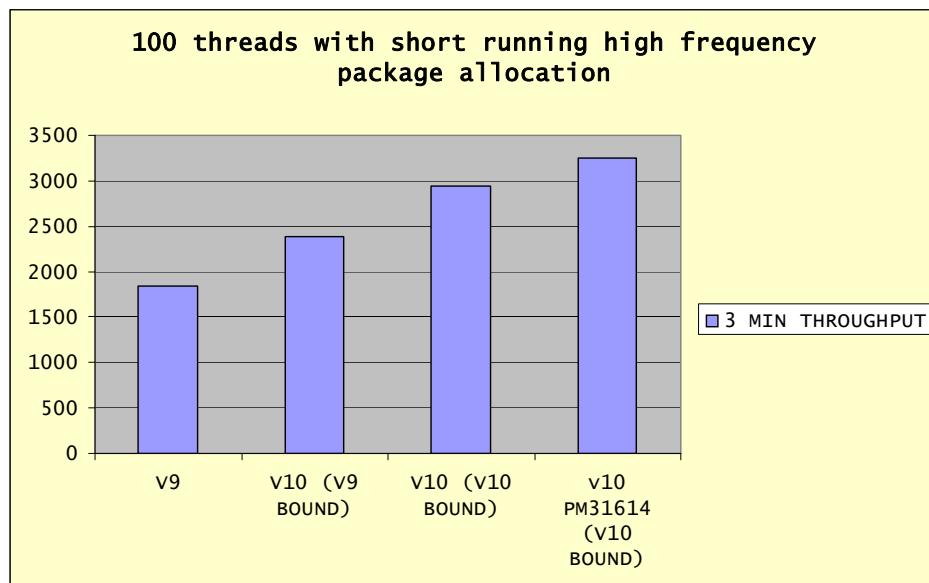


Figure 2-2 Reduction on latch class 24

This sample application with high frequency package allocation has up to 76% increased throughput in DB2 10.

Allocation: APAR PM31614 (PTF UK66374) is a performance improvement for package allocation with DB2 10. With this fix, this application, which is an extreme test case of one SQL SET statement per allocation, there was a total of 75% increased throughput.

2.2.3 Latch class 32

LC32 is a storage manager latch class that latches the storage pool header blocks and 64-bit virtual space allocation blocks. The 64-bit storage manager greatly reduces LC32 contention for shared storage pools above the bar. EDM thread storage pool elimination also removed high frequency allocation and deallocation from EDM fixed storage pools. The change from shared dynamic statement cache pools to thread level storage pools eliminates the need for using LC32.

2.2.4 UTSERIAL elimination

Each DB2 utility job stores its run time information in the DSNDB01.SYSUTILX directory table space. The UTSERIAL lock is an IRLM lock obtained by each utility when it wants to read or write from SYSUTILX. The granularity of this lock is such that it can cause contention when several concurrent utilities are run.

DB2 10 NFM eliminates the UTSERIAL lock and merges tables SYSUTILX and SYSUTIL into one table called SYSUTILX.

DB2 10 now takes a more granular IRLM lock at page level lock against this table to serialize utility access to the DSNDB01.SYSUTILX table space. This lock is a commit-duration lock that is used exclusively by DB2 utilities during utility compatibility checking, with a resource name (dbid.psid) for the database objects that are targeted by the utility. The objective of these locks is to prevent two or more incompatible utilities from serializing successfully when neither has its compatibility information stored in DSNDB01.SYSUTILX yet.

We have executed 20 LOAD jobs concurrently on 2 members with data sharing, both on DB2 9 and DB2 10, 20 table spaces each has 300 partitions and 6 indexes. We also set IRLMRWT and UTIMOUT to the smallest value. On DB2 9, only 7 LOAD jobs completed, whereas 13 LOAD jobs abended with reason code E40085 due to UTSERIAL lock contention. On DB2 10, all 20 LOAD jobs completed successfully. There was no UTSERIAL lock contention.

In this section we have discussed how most of DB2 latches that might impact scalability have an improvement with DB2 10 CM. DB2 10 also reduces catalog contention by eliminating catalog links and converting the catalog tables to use row level locking instead of page-level locking. As a result, DB2 10 shows improvement of performance and increase of transaction throughput.

2.3 Dynamic prefetch enhancements

The following types of DB2 for z/OS prefetch are available:

- ▶ Sequential prefetch: Used for table scans. As soon as the target table space is accessed, two prefetch quantities of sequential pages are read into the buffer pool. Additional requests for a prefetch quantity of pages are issued each time a trigger page is accessed by the executing SQL statement. A trigger page is one that is a multiple of the prefetch quantity. DB2 therefore tries to stay at least one prefetch quantity of pages ahead of the application process. The maximum prefetch quantity is 256 KB or 64 pages for table spaces and indexes that are assigned to 4 KB buffer pools.
- ▶ List prefetch: Used during query execution. RIDs for qualifying rows are obtained from one or more indexes. In most cases, DB2 sorts the RIDs, then issues asynchronous multi-page read requests against the base table space.
- ▶ Dynamic prefetch: Used at query execution. DB2 determines that the pattern of page access for a target table or an index is sequential enough to justify the activation of prefetch processing. If the page access pattern subsequently becomes not sequential enough, prefetch processing is turned off. It is turned on again if sequential enough access resumes.

DB2 10 further enhances prefetch processing, particularly with index access, as follows:

- ▶ Index scan using list prefetch:

DB2 10 can use list prefetch I/O for the leaf pages of an index: this feature greatly mitigates any performance issues of a disorganized index. DB2 10 uses the N-1 level of the index to locate the leaf pages, whereas DB2 9 did not issue getpages for the non-leaf pages when doing an index scan. List prefetch on index leaf pages can greatly reduce the synchronous I/O waits for long running queries accessing disorganized indexes.

Preliminary performance results show a 2 to 6 times elapsed time improvement with simple SQL statements and a small key size using list prefetch compared to synchronous I/Os.

- ▶ Row level sequential detection:

Dynamic prefetch sequential can work poorly when the number of rows per page is large. DB2 10 introduces *row level sequential detection* and unclustered rows are less likely to cause DB2 to fall out of prefetch.

- ▶ Progressive prefetch quantity:

Because row level sequential detection is based on rows, not pages, it is possible to trigger prefetch more quickly, and DB2 will use progressive prefetch quantities. For example, with 4 KB pages, the first prefetch I/O reads 8 pages, then 16 pages, then all subsequent I/Os will prefetch 32 pages. The progressive prefetch quantity applies to indexes and data, even though row level sequential detection does not affect indexes.

These enhancements are available in DB2 10 CM without rebind or bind. See *DB2 10 for z/OS Technical Overview*, SG24-7892 for a description of these topics.

2.3.1 Disorganized index scan using list prefetch

Index-only range scans were measured by varying the percentage of the index that was read. These measurements used a DS8800 with 600 GB 10K RPM drives with IBM FICON® Express 8 channels on a z196 processor. Figure 2-3 shows the results, in elapsed time, of DB2 9 and DB2 10 reading increasing percentages of a disorganized index. In this example the index contains 336,968 leaf pages.

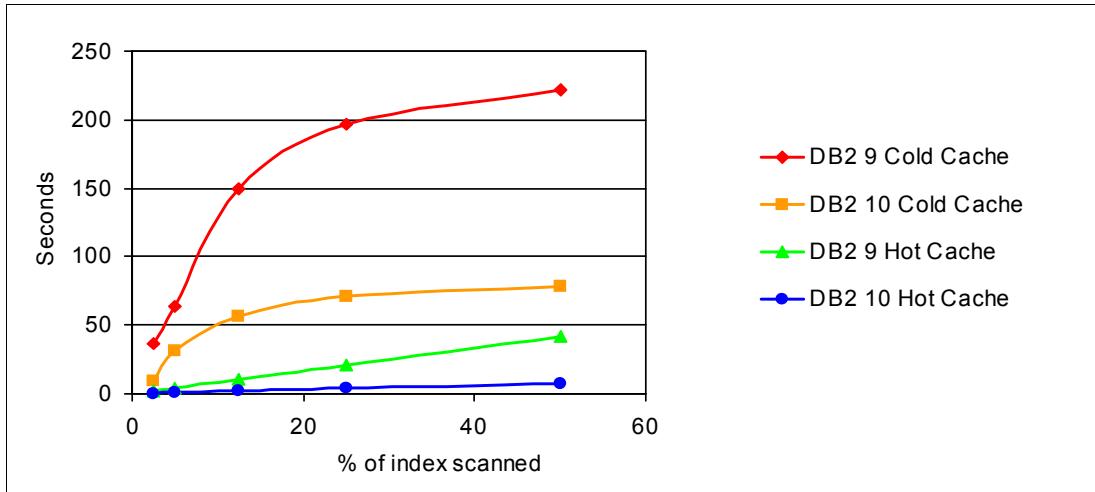


Figure 2-3 Disorganized index scan

A cold cache refers to the situation where the disk controller's cache is cleared before starting the test. When the query reads just 2.5% of the leaf pages, the control unit cache hit ratio is negligible.

After having read more than 10% of the index, we start to get some cache hits, because each cache miss caused twelve 4 KB pages to be read into the cache.

After having read 25% to 30% of the pages, most subsequent I/Os are cache hits, and the performance starts to behave like a *hot cache*.

These tests show that DB2 10 has a higher percentage improvement when we read a small percentage of the index, but when we read more than 30% of the index, we can still get up to three times improvement.

A hot cache refers to the situation where the disk controller's cache is not cleared before the test and the control unit cache hit ratio is 100%, so that the disk is never accessed. With a hot cache, the query response time is a linear function of the percentage of the index scanned and DB2 10 consistently reduces the elapsed time by six times.

The reduction in elapsed time is also complemented with a reduction in CPU time, but there are many factors involved in the process that affect the results, and yours will vary. We used PGFIX(YES) in this test scenario and this option provided a reduction in CPU time. DB2 9 and DB2 10 will show approximately the same amount of page fix time. Because the SRB time is zIIP eligible in DB2 10, you will get an extra advantage in CPU terms if you happen to be executing this operation in an LPAR where zIIP processors are available. This example is based on an index only index scan with no data page access, making the cluster ratio of the index irrelevant for the interpretation of the results.

2.3.2 Row level sequential detection

The sequential detection algorithm that dynamic prefetch used before DB2 10 was always at the page level. When the cluster ratio falls below 100%, the algorithm often broke down when the number of rows per page was large. DB2 10 solves this problem by using row level sequential detection for the data. Row level sequential detection applies to rows, not to indexes. For details, see *DB2 10 for z/OS Technical Overview*, SG24-7892.

We executed a series of tests to show the performance benefits of the changes for queries involving dynamic prefetch as a result of index-to-data range scan access paths. The purpose of the test cases is to compare the behavior of DB2 9 and DB2 10 with increasingly degraded cluster ratio.

The scenario used for these tests was as follows:

1. Initially the table was populated with 20 million rows using inserts with a sequential key. At that point the cluster ratio was 100% and the index was organized.
2. Next a range scan query was executed that read 10% (that is, 2 million) of the rows, using a *cold cache*. Dynamic prefetch was used for both the index and data. This test is called “test case 1.”
3. Then another 200,000 rows were inserted using a random key. All of the new rows were appended to the end of the table because there was no free space. The cluster ratio became 98% and the index began to become disorganized.
4. Then the same range scan was run (reading 1% more rows than test case 1). This was called “test case 2.” The elapsed time for both DB2 9 and DB2 10 increased significantly because of synchronous I/Os for the non-clustered rows, but the number of dynamic prefetch I/Os for the data did not change.
5. After another 200,000 rows were inserted, the same range scan was executed, but this time the number of dynamic prefetch I/Os for the data with DB2 9 began to drop. With DB2 10, the number of dynamic prefetch I/Os held constant, thanks to row level sequential detection.

For this scenario, we read 10% of the rows in key sequential order. The row size was 49 bytes and the page size was 4 KB (81 rows per page.)

Table 2-1 shows a summary of the test scenario.

Table 2-1 Test scenario for prefetch performance reports - Row size 49 bytes

Test case	Cluster ratio	Cardinality	NPAGES
1	100%	20,000,000	253,167
2	98%	20,200,000	256,024
3	96%	20,400,000	258,882
4	94%	20,600,000	261,740
5	92%	20,800,000	264,598

Figure 2-4 shows elapsed time and dynamic prefetch results. When the cluster ratio reached 94%, DB2 9 stopped using dynamic prefetch altogether, but DB2 10 never stopped using dynamic prefetch. When the cluster ratio was 92%, DB2 10 reduced the elapsed time by about 45%.

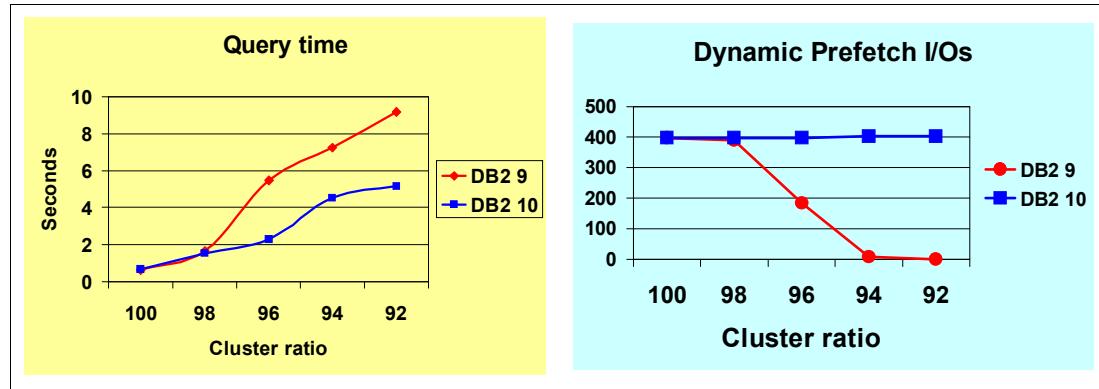


Figure 2-4 Total elapsed time and dynamic prefetch I/O versus cluster ratio

The same tests as before were repeated in order to evaluate the impact of larger rows. The row size was doubled, so that there were only 40 rows per 4 KB page.

Table 2-2 shows a summary of the test scenario.

Table 2-2 Test scenario for prefetch performance reports - Row size 98 bytes

Test case	Cluster ratio	Cardinality	NPAGES
1	100%	20,000,000	500,000
2	96%	20,400,000	520,000
3	92%	20,800,000	540,000
4	88%	21,200,000	560,000
5	84%	21,600,000	580,000

Figure 2-5 illustrates that with fewer rows per page, dynamic prefetch performed better in DB2 9, because the number of dynamic prefetches did not begin to decrease until the cluster ratio fell below 96%.

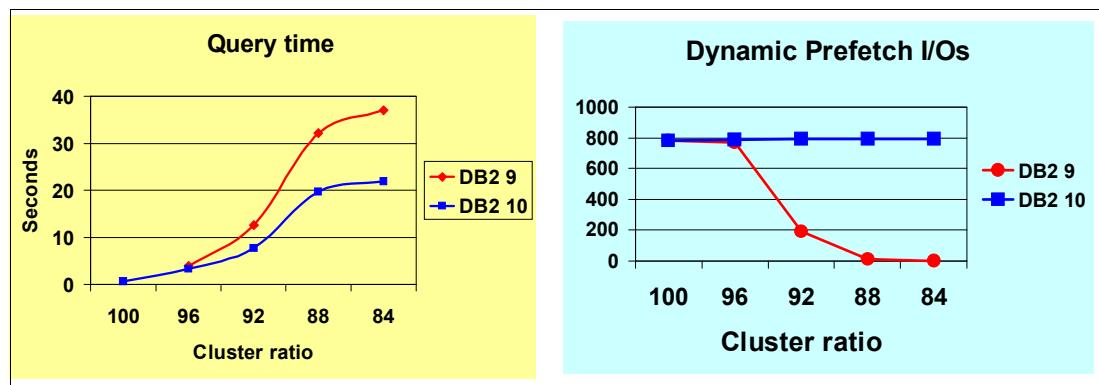


Figure 2-5 Row level sequential detection with larger row

If RUNSTATS is run often enough so that the cluster ratio metric is kept current, DB2 will more likely choose list prefetch over dynamic prefetch when the cluster ratio falls below 90%. In other words, the intended operating range for dynamic prefetch is for cluster ratios above 90%. When the number of rows per page is small, DB2 9 did all right in this range, but it did not do well when the number of rows per page was large. DB2 10 helps with these cases.

Because these tests read 10% of the rows using a cold cache, as well as a cold buffer pool, the non-clustered rows caused significant performance problems. However, if the query were to scan 50% of the rows, for example, both the DB2 buffer hit ratio and the cache hit ratio will increase significantly, in which case the non-clustered rows will not add much more to the elapsed time, and the query will start to become more CPU bound.

Thus, it is the shorter queries for which DB2 9 has the most trouble and where DB2 10 helps the most.

For additional considerations and measured performance results, see 3.4, “Disk storage enhancements” on page 65 where we describe DS8800 prefetch.

2.3.3 Progressive prefetch quantity

Because DB2 10 introduces row level sequential detection and thus triggers prefetch more quickly (after a minimum of 5 rows, as opposed to 5 pages in DB2 9), dynamic prefetch in DB2 10 also uses a progressive prefetch quantity. For example, with 4 KB pages the first prefetch I/O reads 8 pages, then 16 pages, then all subsequent I/Os will prefetch 32 pages (as in DB2 9).

While row level sequential detection applies only to data, a progressive dynamic prefetch quantity is used for both index and data.

2.3.4 Summary on prefetch improvements

Unclustered rows are less likely to cause DB2 to stop prefetch. DB2 prefetch works like DB2 9 at very high cluster ratios, such as over 98%, and improves elapsed time substantially as the cluster ratio is reduced.

The introduction of list prefetch for index scan access path can improve performance when accessing indexes that are not perfectly organized. Measurement showed up to 6 times elapsed time reduction when performance is compared to DB2 9.

DB2 10 provides important performance enhancements for dynamic prefetch. Performance measurements of SQL statements involving dynamic prefetch shows that DB2 10 row level sequential detection preserves good sequential performance for the clustered pages.

Both list prefetch for index scan and row level sequential detection can relieve the need for REORG utilities because this provides an improved tolerance for disorganized indexes and data and therefore reduces the frequency of reorganization.

2.4 Buffer pool enhancements

The buffer pool enhancements in DB2 10, which are all available in CM, allow you to increase transaction throughput and take advantage of larger buffer pools by reducing latch class 14 and 24 contention, reducing buffer pool CPU overhead, and avoiding transaction I/O delays by preloading objects into the buffer pool.

We describe these enhancements in the following sections:

- ▶ Buffer storage allocation
- ▶ In-memory table spaces and indexes

2.4.1 Buffer storage allocation

In previous versions of DB2, storage is allocated for the entire size of the buffer pool (VPSIZE) when the buffer pool is first allocated, (the first logical open of a page set), even if no data was accessed in any table space or index using that buffer pool. Now, buffer pool storage is allocated on-demand as data is brought in. If a query touches only a few data pages, only a small amount of buffer pool storage is allocated.

Here, *logical open* is when the page set is either physically opened (the first SELECT) or pseudo opened (the first UPDATE after being physically opened for read). There is no buffer pool allocation as it is already allocated at read time.

In addition, for a query that performs index-only access, the buffer pool for the table space does not need to have any buffer pool storage allocated. DB2 10 no longer performs logical open of the table space page set for index-only access, thereby avoiding table space buffer pool allocation as well as open page set lock and unlock for the table space.

For buffer pools defined with PGFIX=YES, DB2 requests buffer pools to be allocated using 1 MB page frames if they are available, rather than 4 KB pages frames. 1 MB page frames are available for z10 and later models. You define the number of 1 MB page frames that are available to z/OS in the LFAREA parameter of SYS1.PARMLIB(IEASY\$xx). Manipulating storage in 1 MB chunks rather than 4 KB chunks can significantly reduce CPU overhead in memory management, by increasing the hit ratios of the hardware translation look-aside buffer.

Note that although DB2 can request 1 MB page frames from the operating system, DB2 itself still manages the buffer pools as 4 KB, 8 KB, 16 KB, and 32 KB pages. Nothing changes in DB2 for table space page sizes.

DB2 requests 1 MB page frames only for PGFIX=YES buffer pools. If there are no more 1 MB page frames available, then DB2 requests 4 KB page frames. Buffer pools do not automatically shrink when page sets are logically closed. When allocated, the buffer pools remain until either all the page sets are physically closed or DB2 is stopped. Workload managed buffer pool support introduced in DB2 9 can dynamically reduce the size of buffer pools when there is less demand. See *DB2 9 for z/OS Performance Topics*, SG24-7473 for details about WLM buffer pool support.

There can be a temptation either to make buffer pools bigger than they normally might be or to define more buffer pools with PAGEFIX=YES, thinking that the extra buffer pool space might not be used because it is allocated only when it is needed. We advise that you resist this temptation. You still need enough real storage to back the buffer pools to keep real paging at an acceptable level and your amount of real storage has not changed.

For BP0, BP8K0, BP16K0, and BP32K, the minimum size set by DB2 is 8 MB.

2.4.2 In-memory table spaces and indexes

In the past, the cost of physically opening a page set was born by the first SQL statement to access that data set. This cost adversely impacted application performance, typically after DB2 restart. In addition, some tables might be critical for application performance, so they need to be always resident in the buffer pools.

The two major advantages of an in-memory buffer pool are that DB2 does not need to schedule a prefetch engine anymore because the pages are expected to be in memory anyway, and DB2 can save the cost of the LRU buffer management. DB2 9 already provides these advantages if you specify VPSEQT(0), but PGSTEAL(NONE) goes one step further by loading the objects sequentially. If the objects were going to remain in memory, how it gets loaded into the buffer pool will not matter much, but one of the problems with VPSEQT(0) is that when DB2 is restarted, the objects get loaded using synch I/O, which is exactly the problem that in-memory objects must avoid.

DB2 10 provides a buffer pool attribute that you can use to specify that all objects using that buffer pool are in-memory objects. The data for in-memory objects is preloaded into the buffer pool at the time the object is physically opened, unless the object has been opened for utility access. The pages remain resident as long as the object remains open.

When the page set is first accessed (the first getpage request initiated by the SQL statement) an asynchronous task is scheduled under the DBM1 address space to prefetch the entire page set into the buffer pool. The CPU for loading the page set into the buffer pool is therefore charged to DB2. If this first getpage happens to be for a page that is being asynchronously read at the time, then it waits. Otherwise, the requested page is read synchronously.

If a page set is opened as a part of DB2 restart processing, the entire index or table space is not prefetched into the buffer pool.

Page sets can still be physically opened before the first SQL access by using the -ACCESS DATABASE command introduced in DB2 9. Now, you can also preload all of the data into the buffer pool or pools before the first SQL access.

To realize the benefit of in-memory page sets, you still need to make sure that the buffer pools are large enough to fit all the pages of all the open page sets. Otherwise, I/O delays can occur as the buffer pool fills up and DB2 must steal buffers, (on a FIFO in this case). This behavior is the same as with previous versions of DB2.

In-memory page sets help DB2 to reduce overall least recently used (LRU) chain maintenance and latch class 14 contention, because there is much less buffer pool activity and buffer pools with in-memory page sets are managed with FIFO. They also avoid unnecessary prefetch and latch class 24 contention, because the data is already in the buffer pools and because prefetch is disabled for in-memory page sets.

Unlike previous solutions to in-memory page sets where you preloaded data by running an SQL statement, the optimizer takes into consideration that the data is preloaded. So, access paths can be different when accessing in-memory page sets; however, this access path is not attributed to in-memory page in the PLAN_TABLE.

Important: It is more important in DB2 10 to make sure that you have large enough buffer pools to store all the in-memory page sets. If the buffer pools are too small to store all of the data, then performance can be impacted, because DB2 might be using a non optimal access plan that did not allow for the extra I/O. In addition, if DB2 needs to perform I/O to bring a page into the buffer pool for processing, this I/O is synchronous because prefetch is disabled.

The new option available for the PGSTEAL parameter of the -ALTER BUFFERPOOL command, PGSTEAL(NONE), indicates that no page stealing can occur. All the data that is brought into the buffer pool remains resident. Figure 2-6 shows the new syntax.

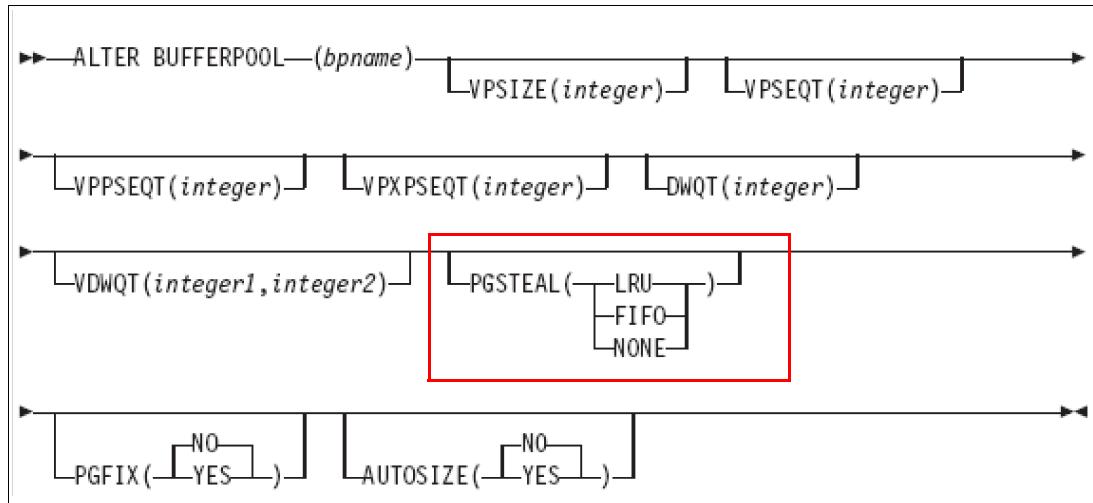


Figure 2-6 ALTER BUFFER POOL PGSTEAL syntax

Altering the PGSTEAL value takes effect immediately. For PGSTEAL LRU or FIFO, new pages added to the LRU chain take the new behavior immediately, but the ALTER does not affect the pages already on the chain. Altering the buffer pool to PGSTEAL(NONE) also has an immediate effect. The ALTER schedules prefetches for all of the page sets in the buffer pool.

You can define in-memory table spaces and indexes in DB2 10 CM. On fallback to DB2 9, PGSTEAL=NONE reverts to its previous value, which is LRU if the parameter was never changed. On remigration, PGSTEAL returns to NONE if it was set prior to fallback.

The following buffer manager display messages are modified to accommodate PGSTEAL=NONE:

DSNB4021

The BUFFERS ACTIVE count is removed, because it reflects the number of buffers that have ever been accessed, which is essentially the same behavior as BUFFERS ALLOCATED:

DSNB406I
DSNB519I

IFCID 201 records the buffer pool attributes changed by the -ALTER BUFFERPOOL command. A new value of *N* indicates that PGSTEAL(NONE) is defined for the QW0201OK and QW0201NK trace fields, which records the old and new values of PGSTEAL respectively. Similarly, IFCID 202, which records the current attributes of a buffer pool, also uses *N* to indicate PGSTEAL(NONE).

2.4.3 DB2 10 buffer pool prefetch and deferred write activities

Buffer pool prefetch, which includes dynamic prefetch, list prefetch, and sequential prefetch activities, is 100% zIIP eligible in DB2 10. DB2 10 zIIP eligible buffer pool prefetch activities are asynchronously initiated by the database manager address space (DBM1) and are executed in a dependent enclave that is owned by the system services address space (MSTR). Because asynchronous services buffer pool prefetch activities are not accounted to the DB2 client, they show up in the DB2 statistics report instead. Deferred write is also eligible for zIIP.

With APAR PM30468 (PTF UK64423), prefetch and deferred write CPU, when running on a zIIP processor, are reported by WLM under the DBM1 address space, not under the MSTR.

In 3.2.2, “Asynchronous I/O zIIP eligibility” on page 56, we provide an example of exploiting WLM Report Classes and RMF reports for the observation of the zIIP CPU time on prefetch operations.

2.5 Work file enhancements

In this section, we discuss the following work file management changes:

- ▶ Support for spanned work file records
- ▶ In-memory work file enhancements

2.5.1 Support for spanned work file records

DB2 10 allows work file records to be spanned, which provides the functionality to allow the work file record length to be up to 65529 bytes by allowing the record to span multiple pages. This support alleviates the issue of applications receiving SQLCODE -670 (SQLSTATE 54010) if the row length in the result of a join or the row length of a large sort record exceeds the 32 KB maximum page size of a work file table space.

The spanned work file records support allows the records of the work files created for JOINs and large sorts to span multiple pages to accommodate larger record length and larger sort key length for sort records.

When the row length in the result of a JOIN or the row length of a large sort record exceeds approximately one page of work file space, the work file manager allows the work file record to span multiple pages, provided that the work file record length is within the new limit of 65529 bytes.

The *maximum* limit for sort key length for sort is increased from 16000 to 32000 bytes. This limit alleviates the issue of applications receiving SQLCODE -136 (SQLSTATE 54005) if the length of the sort key derived from GROUP BY, ORDER BY, DISTINCT specifications in the SQL statement exceeds the limit of 16000 bytes.

The (SQLCODE -670,SQLSTATE 54010) or (SQLCODE -136,SQLSTATE 54005) is issued when the row length in the result of a JOIN or the row length of a large sort record exceeds the limit of 65529 bytes or when the sort key length for a sort exceeds the limit of 32000 bytes, respectively. Spanned records support is available in DB2 10 new-function mode.

2.5.2 In-memory work file enhancements

The in-memory work file enhancements are intended to provide performance improvements to help workloads with queries that require the use of small work files to consume less CPU time. These enhancements facilitate wider usage of the in-memory work files above the 2 GB bar by allowing simple predicate evaluation for work files. This support is intended to reduce the CPU time consumed by workloads that execute queries that require the use of small work files. In-memory work file support is available in DB2 10 conversion mode.

In DB2 9, the real time statistics table SYSIBM.SYSTABLESTATS maintains the disk storage allocated information for each table space in the work file database by means of one row per table space. In DB2 10, for partition-by-growth table spaces in the work file database, there is one row for each partition of the table space.

If you have programs to monitor total disk storage that is used for the work file database, you might need to change the programs to adapt to the partition-by-growth statistics. The DB2 10 data collected from the DB2 accounting trace class (1,2) and examination of the fields Accounting class 2 elapsed time and Accounting class 2 CPU time can be used for comparison with corresponding baseline data from prior releases.

Reference: See *DB2 10 for z/OS Technical Overview*, SG24-7892 for details on installation changes related to work files.

DB2 10 uses in-memory work files in the following situations:

- ▶ Last sort in a top query block with ORDER BY or GROUP BY clause and the size of the sort records is less than 1 million bytes. For example:
 - SELECT c1, c2 FROM T1 ORDER BY c1;
 - SELECT MAX(c1), c2 FROM T1 GROUP BY c2;
- ▶ Sort for join operations and the sort record length < 1000 bytes, the number of rows < 255, and the size of the sort records < 32 KB

Performance result will vary depending on the type of query, but performance observations can be summarized as follows:

- ▶ Up to 50% CPU time improvement for last sort in a top query block
- ▶ Up to 2% CPU time improvement for sort for join operations

Improvements are more noticeable in a multi threaded application due to the reduction of spacemap page contentions. The DB2 utilization of in-memory work files shows as a reduction of getpages in the work file buffer pool, and it can be monitored by the exploitation of new counters in IFCID 2.

Attention: An in-memory work file used for the last sort in a top query block is disabled if:

- ▶ Cursors are defined with hold.
- ▶ Parallelism is turned on.
- ▶ Set function work file scan is on.

Other sort enhancements provided by DB2 10 include these:

- ▶ Increased the default sort pool storage to 1 MB
- ▶ Implemented a hash technique for GROUP BY queries:
 - Up to 18% CPU time decrease occurred for an individual query in lab query workload.
 - 12 out of 87 queries had > 3% CPU time decrease.
 - 10 out of 87 queries had > 3% CPU time increase, but they are short running queries.
- ▶ Implemented a hash technique for sparse index to help improve probing.
- ▶ Removed padding of variable length data fields during the input phase:
 - Up to 1.5% CPU time decrease occurred.
- ▶ Improved FETCH FIRST n ROWS processing if sort record size < 128 KB for both ORDER BY and GROUP BY and when the sort record size > 128 KB, only the n rows to be fetched are written to the work files.

The sample query used for the GROUP BY performance measurements is shown in Example 2-1.

Example 2-1 Sample query for GROUP BY performance measurements

```
SELECT CHAR8, CHAR14N, CHAR1,  
SUM (INT1)  
    FROM TBSRTA00  
   WHERE CHAR14 < 'P00200001ARTNM'  
GROUP BY  
CHAR8, CHAR14N, CHAR1
```

You can exploit the performance IFCID 95 and 96 to check what kind of GROUP BY sort algorithm is used. IFCID 95 and 96 can be used to see sort information at the SQL level. This includes information about sort keys, number of runs, and duration of sorts. It is generally a low-overhead trace that provides useful information for analyzing DB2 SQL statements that can result in sort processing.

Example 2-2 shows the OMEGAMON PE syntax that can be used for creating a report of these IFCIDs.

Example 2-2 OMEGAMON PE syntax for reporting IFCIDs 95 and 96

```
GLOBAL TIMEZONE(+07:00)  
    PAGESIZE(66)  
    DB2PM  
        RECTRACE  
            TRACE LEVEL(LONG)  
            INCLUDE(IFCID(95,96))
```

Example 2-2 shows an extract of a record trace report showing IFCIDs 95 and 96.

Example 2-3 OMEGAMON PE record trace report for IFCIDs 95 and 96

```
1 LOCATION: DB0B          OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1)      PAGE: 1-1  
    GROUP: N/P  
    MEMBER: N/P  
    SUBSYSTEM: DB0B  
    DB2 VERSION: V10  
OPRIMAUTH CONNECT INSTANCE END_USER WS_NAME          TRANSACT  
ORIGAUTH CORRNAME CONNTYPE RECORD TIME DESTNO ACE IFC DESCRIPTION DATA  
PLANNNAME CORRNMBR          TCB CPU TIME           ID  
-----  
SYSADM BATCH C77DA8AE3672 'BLANK' 'BLANK'          'BLANK'  
SYSADM GRB0A01A TSO 16:37:17.26386409 258472 1 95 SORT START --> NETWORKID: USIBMSY LUNAME: APPLDB0B LUWSEQ: 1  
DSNTEPA1 'BLANK'          N/P  
                           16:37:21.02280965 258473 1 96 SORT END <-> 'BLANK'  
                           N/P  
                           NETWORKID: USIBMSY LUNAME: APPLDB0B LUWSEQ: 1  
                           RECN0 TO BE DONE AREA 29 KEYSZ 24  
                           SIZE 53 WORK 5 RET 0  
                           IWORK 4 ROW DELTO BE DONE PASSES 1  
                           TYPE ESAG STMTNO 1928 WORKFILES 0  
                           COLLECTION ID DSNTEP2  
                           PROGRAM NAME DSN@EP2L  
                           SORT COLUMNS 5 SORT KEYS 3  
                           PARTITIONING BY SORT: NO SORT IN ADDITION: NO  
                           PARTITIONING OCCURRED: N  
                           QW0096IN TO BE DONE QW0096RD TO BE DONE  
                           QW0096RU 0  
1 LOCATION: DB0B          OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1)      PAGE: 1-2  
    GROUP: N/P  
    MEMBER: N/P  
    SUBSYSTEM: DB0B  
    DB2 VERSION: V10  
0 ACE          ACE          ACE          ACE          ACE          ACE          ACE          ACE          ACE          ACE  
NUMBER ADDRESS NUMBER ADDRESS NUMBER ADDRESS NUMBER ADDRESS NUMBER ADDRESS  
-----  
1 X'1D263C00'  
ORECORD TRACE COMPLETE
```

The TYPE section of this report indicates the type of SORT and might show these options:

- ▶ ESAG: Group by using HASH sort
- ▶ RCYC: Group by using recycle sort
- ▶ ESAT: Tag sort
- ▶ ESA: All other sorts
- ▶ NONE: No sort occurred

ESAG indicates the use of the HASH technique, and this SORT section of the query gets the benefits of DB2 10 for GROUP BY statements.

2.5.3 Work file table spaces

For optimal sort work performance, IBM has always advised to allocate many small work file table spaces, because sort work involves a lot of I/O parallelism. One way to force DB2 to spread a sort work file across multiple table spaces is to set SECQTY to 0 for every table space. However, a declared global temporary table (DGTT) cannot span multiple table spaces, and will be limited in size if SECQTY is set to 0.

Prior to DB2 9, a natural separation existed between these two groups of functions, sort and DGTT, because DGTTs were contained in the TEMP database and work files were contained in the WORKFILE database.

In DB2 9, the TEMP database was removed and users were not able to limit space usage effectively between DGTT and work files (sort work files, created global temporary tables (CGTT), trigger transition tables, and so on). Because sorts tend to be large, they might use most of the space, and if a DGTT also happens to use the same table space, it can have space problems because it is limited to a single table space.

Consequently, DB2 9 introduced APAR PK70060 to create an initial preference scheme based on SECQTY allocation.

Then DB2 9 APAR PM02528 introduced a DSNZPARM in DSN6SPRM called WFDBSEP, which specifies whether DB2 must provide an unconditional separation of table spaces in the WORKFILE database based on the table spaces' allocation attributes. The DSNZPARM might be suitable for users of the WORKFILE database who prefer to set aside a fixed amount of space for DGTT work versus work file work, and do not want it to be exceeded, even at the cost of potentially failing workloads (for example, DSNT501I / SQLCODE -904 due to lack of space) until more space is manually added. In essence, WFDBSEP=YES separates DGTTs and work files similarly to V8, where they were serviced in two separate databases (TEMP and WORKFILE). For WFDBSEP, the following considerations exist:

- ▶ If the value is YES, DB2 always directs DGTT work only to DB2-managed (STOGROUP) work file table spaces defined with a non-zero SECQTY and work file work only to other work file table spaces (DB2-managed table spaces defined with a zero SECQTY or user-managed table spaces). If no table space with the preferred allocation type is available, DB2 issues an error (such as message DSNT501I and/or SQLCODE -904).
- ▶ If the value is NO, DB2 attempts to direct declared global temporary table (DGTT) work to DB2-managed (STOGROUP) work file table spaces defined with a non-zero SECQTY and work file work to any other work file table space (DB2-managed table spaces defined with a zero SECQTY or user-managed table spaces). If no table space with the preferred allocation type is available, DB2 selects a table space with a non-preferred allocation type. In DB2 10, the WFDBSEP subsystem parameter specifies whether DB2 must provide an unconditional separation of table spaces in the work file database based on the allocation attributes of the table spaces. The default value for WFDBSEP is NO.

Figure 2-7 shows option 10 of panel DSNTIPB. This option allows you to define the values for WFDBSEP, which is part of the DSN6SPRM macro. See the *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974 for details.

You can use option 9, MAXTEMPS, to limit the maximum space a single task can allocate.

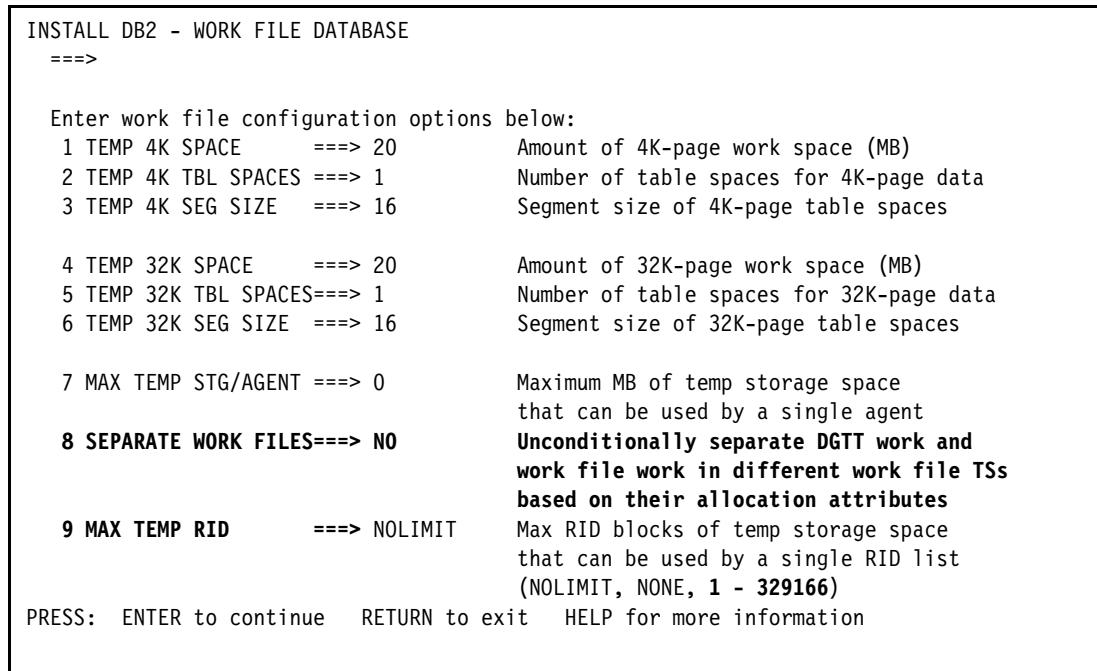


Figure 2-7 DB2 installation panel DSNTIPB showing option 10 WFDBSEP

There are, however, other possible storage related problems, such as these:

- ▶ With system managed storage, extent consolidation makes it impossible to prevent a DGTT from growing to 64 GB unless the secondary space quantity is 0.
- ▶ Specifying a small secondary quantity can result in a small fragmented segmented table space.

In order to better address these issues, DB2 10 provides another solution. In addition to classic segment table spaces, DB2 10 NFM allows work files to be defined as partition by growth, or PBG, where the user is able to control the physical storage growth by specifying MAXPARTITIONS and DSSIZE in the DDL. They are the preferred table space for DGTT. For those customers who want to restrict the size of a DGTT below 64 GB, they can now specify a small DSSIZE with MAXPARTITIONS 1. On the other hand, a customer can now generate DGTTs bigger than 64 GB by specifying a value for MAXPARTITIONS greater than 1. Furthermore, if the customer wants the DGTT limit to remain 64 GB as in DB2 9, this can be done using a single PBG data set (and possibly a single extent) rather than allocating 32 different data sets of 2 GB each.

It is also possible to exploit DGTT bigger than 64 GB (DGTT cannot span multiple work file table spaces). Considering that some users prefer to avoid multi-volume data sets, the combination of MAXPARTITIONS and DSSIZE enables you to choose parameter values according the volume sizes available on the system that will minimize the likelihood of needing to use multi-volume data sets.

Here we summarize these considerations:

- ▶ For DGTT usage, DB2 picks PBG table space as the first priority. If PBG is not available, DB2 picks work file table spaces with non zero SECQTY datasets.
- ▶ For sort work files, DB2 prefers zero SECQTY datasets or user defined datasets for sort work files.

The measurements have shown that both performance and physical storage are comparable for concurrent DGTT (create/insert/select/delete) between PBG and segmented table space, and physical storage is comparable for DGTTs using segmented table space between DB2 9 and DB2 10.

We can list the following considerations of usage:

- ▶ Create PBG table spaces for DGTTs. Only if they are not available, DB2 will look for segmented table spaces with non-zero SECQTY.
- ▶ Create segmented table space with 0 SECQTY for sort.
- ▶ Set WFDBSEP = YES for DB2 managed or user managed table spaces to segregate DGTTs and sort work file usage.

2.6 Logging enhancements

DB2 10 includes the following enhancements, all active in CM, that reduce logging delays:

- ▶ Log latch contention reduction
- ▶ Long term page fix log buffers
- ▶ LOG I/O enhancements

2.6.1 Log latch contention reduction

Since DB2 Version 1, DB2 has used a single latch for the entire DB2 subsystem to serialize updates to the log buffers when a log record needs to be created. Basically, the latch is obtained, an RBA range is allocated, the log record is moved into the log buffer, and then the latch is released. DB2 10 makes several changes to the way this latch process works, which increase logging throughput significantly and reduce latch class 19 contention. The changes improve the latch management and reduce the time that the latch is held.

2.6.2 Long term page fix log buffers

DB2 10 page fixes the log buffers permanently in memory. At DB2 start, all the buffers as specified by OUTBUFF are page fixed.

The OUTPUT BUFFER field of installation panel DSNTIPL allows you to specify the size of the output buffer that is used for writing active log data sets. The maximum size of this buffer is 400,000 KB, the default is 4,000 KB (4 MB), and the minimum is 400 KB.

Generally, the default value is sufficient for good write performance. Increasing OUTBUFF beyond the DB2 10 default might improve log read performance. For example, ROLLBACK and RECOVER with the new BACKOUT option can benefit by finding more data in the buffers. COPY with the CONSISTENT option might benefit too.

Whether a large OUTBUFF is desirable depends on the trade-off between log read performance (especially in case of a long running transaction failing to COMMIT) versus real storage consumption. Review your OUTBUFF parameter to ensure that it is set to a realistic trade-off value. The QJSTWTB block in the QJST section of IFCID 001 can indicate if the log buffer is too small. This counter represents the number of times that a log record write request waits because there are not available log buffers.

Important: In DB2 10, log buffers are permanently page-fixed. When you estimate real storage usage, you must use the entire size that you specify for the OUTBUFF parameter. To avoid page-fixing more storage than necessary, carefully choose the setting for OUTBUFF.

2.6.3 LOG I/O enhancements

In DB2 9, if a COMMIT needs to rewrite page 10, along with pages 11, 12, and 13, to the DB2 log, DB2 first serially writes page 10 to log 1, then serially writes page 10 to log 2, and then writes pages 11 through 13 to log 1 and log 2 in parallel. In effect, DB2 9 does four I/Os and waits for the duration of three I/Os.

The first time a log control interval is written to disk, the write I/Os to the log data sets are performed in parallel. However, if the same 4 KB log control interval is again written to disk, the write I/Os to the log data sets are done serially to prevent any possibility of losing log data in case of I/O errors on both copies simultaneously.

Figure 2-8 shows a graphical representation of the synchronous I/O involved in DB2 9 at COMMIT time. Assume that the log control interval is rewritten during the COMMIT. This example shows a log page partially used, but the I/O rewrites a complete 4 KB log page. The number 1 represents the I/O required for writing the log page on Log 1. DB2 9 waits for this I/O to be completed before starting the I/O on Log 2, represented by the number 2 in this figure.

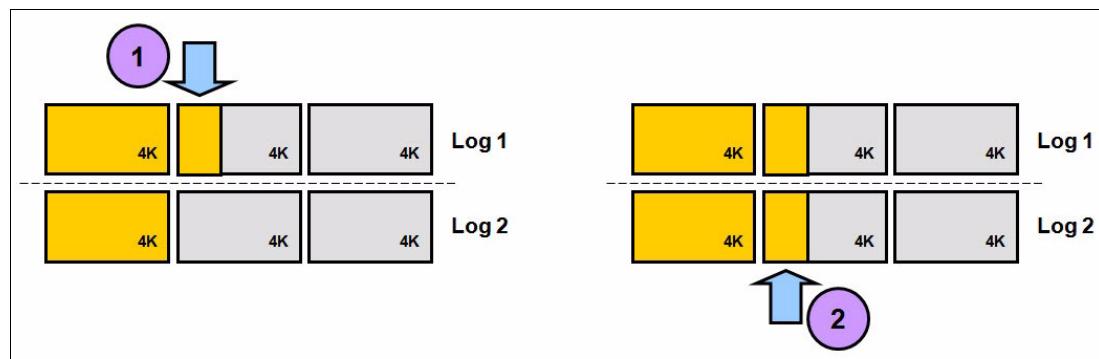


Figure 2-8 DB2 9 COMMIT synchronous I/O

Current DASD technology writes I/Os to a new area of DASD cache each time rather than disk. There is no possibility of a log page being corrupted when it is being re-written. DB2 10 takes advantage of the non-volatile cache architecture of the I/O subsystem. DB2 10 simply writes all four pages to log 1 and log 2 in parallel. Hence, DB2 10 writes these pages in only two I/Os, and waits for the duration of only one I/O (that is, whichever of the two I/Os takes the longer.)

DB2 rewrites the page asynchronously to both active log data sets. In this example, DB2 chains the write for page 10 with the write requests for pages 11, 12, and 13. Thus, DB2 10 reduces the number of log I/Os and improves the I/O overlap.

Figure 2-9 shows a graphical representation of the synchronous I/O involved in DB2 10 at COMMIT time, assuming the same conditions as in Figure 2-8 on page 40. The number 1 represents the I/O required for writing the log page on Log 1. DB2 10 does not wait for this I/O to be completed before starting the I/O on Log 2, represented by the number 2. Assuming that the elapsed time for writing two log pages is the same as writing a single page, you can expect a dramatic reduction in log suspension time for these types of operations.

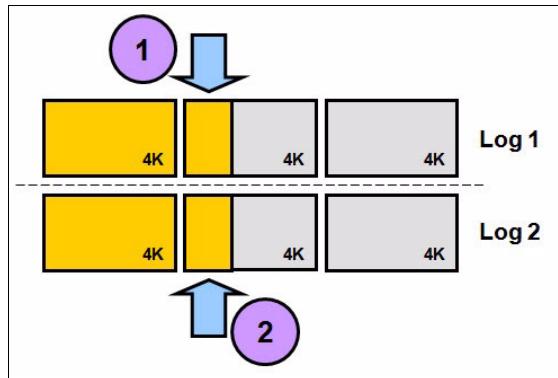


Figure 2-9 DB2 10 COMMIT synchronous I/O

2.6.4 Performance with log writes

Log writes are divided into two categories: asynchronous and synchronous.

- ▶ Synchronous writes

Synchronous writes usually occur at commit time when an application has updated data. This write is called 'forcing' the log because the application must wait for DB2 to force the log buffers to disk before control is returned to the application. If the log data set is not busy, all log buffers are written to disk. If the log data set is busy, the requests are queued until it is freed.

- ▶ Asynchronous writes

Asynchronous writes are typical of batch insert and update jobs. These asynchronous writes occur when 20 log buffers have been accumulated.

Log synchronous writes performance

Figure 2-10 shows a comparison between DB2 9 and DB2 10 of the suspension time per commit. This chart shows the difference in performance, reported as suspension time per commit, for commit operation up to 3 pages per commit.

The logging improvements of DB2 10 are evident in this chart when compared to DB2 9. DB2 logging suspension time drops by about 50%.

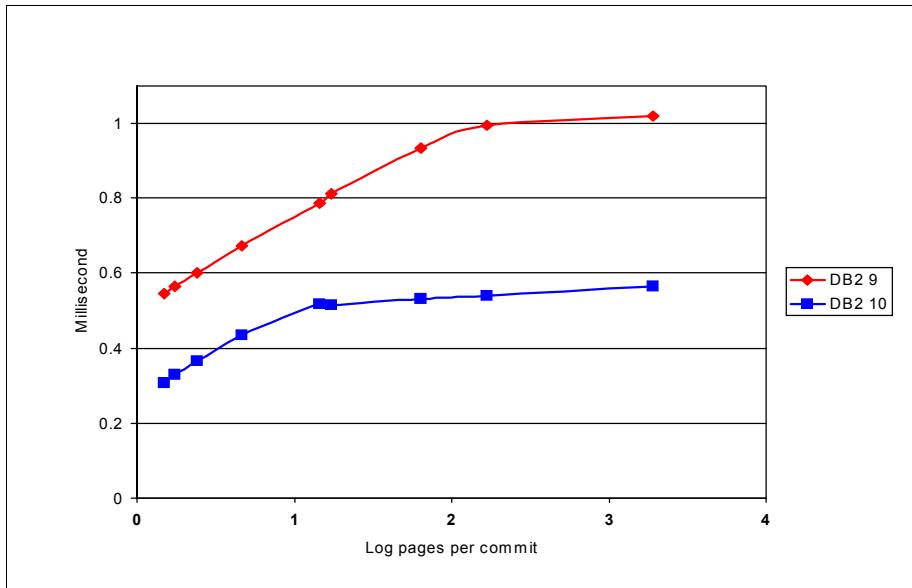


Figure 2-10 Suspension time per commit

Figure 2-11 shows a comparison between DB2 9 and DB2 10 of the suspension time per commit for an extended range of log pages per commit. In this case the tests were extended to more than 25 pages per commit. This shows that the advantage of DB2 10 compared to DB2 9 is maintained for that range.

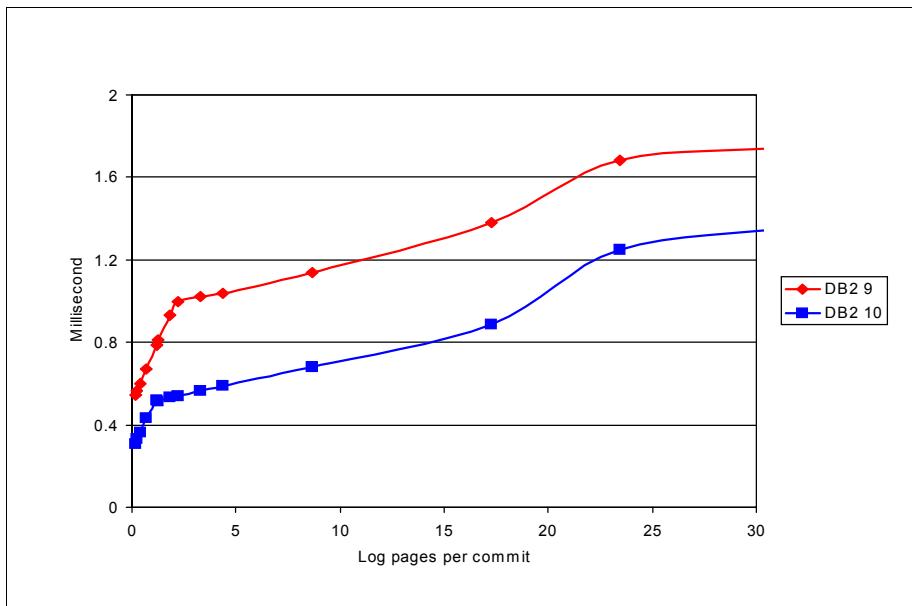


Figure 2-11 Suspension time per commit

The commit response time beyond 20 pages is actually much more variable than what is shown in Figure 2-11. In this example, an asynchronous I/O for 20 pages was started prior to the COMMIT, but it had not completed yet. Thus, the COMMIT had to wait for the asynchronous I/O to complete before starting the synchronous I/O.

Log asynchronous writes performance

To report the maximum logging asynchronous throughput, we executed a series of tests under the following conditions: We allocated one big data set extent and we preformatted it using the load utility. In order to make sure that deferred writes will not affect the performance and the consistency of the measurements, we allocated a buffer pool that was bigger than the amount of data that was going to be inserted, and we set VDWQT to 90%.

We used a large row size in order to have one row per page to minimize the CPU time. We created no indexes and we issued COMMITS very infrequently so the CPU time was negligible and all of the suspension time was *other log write I/O*. During the tests, we observed the log1 and log2 devices very close to 100% busy.

The measurements were done on a System z z196 machine. We used both a DS8300 disk controller with FICON Express 4 and a DS8800 with FICON Express 8.

Figure 2-12 shows the observed results when DB2 10 and DB2 9 throughput is compared. This test scenario is probably not a representation of most of today's business processes, but it helps to visualize the performance improvements of DB2 10 as well as the value of the DS8800 and High Performance FICON.

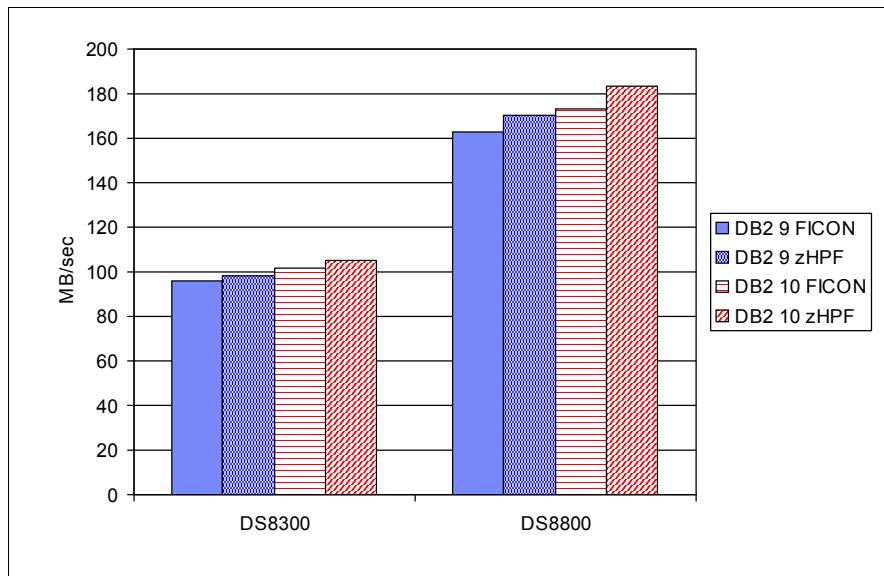


Figure 2-12 Maximum DB2 log throughput

For more information about disk storage exploitation, see 3.4, "Disk storage enhancements" on page 65.

2.7 I/O parallelism for index updates

When DB2 9 inserts a row into a table, it must perform a corresponding insert into all the indexes that are defined on that table. All of these inserts into the indexes are done serially, one at a time.

DB2 10 provides the ability to insert into multiple indexes that are defined on the same table in parallel. Index insert I/O parallelism manages concurrent I/O requests on different indexes into the buffer pool in parallel, with the intent of overlapping the synchronous I/O wait time for different indexes on the same table. This processing can significantly improve the

performance of I/O bound insert workloads. It can also reduce the elapsed times of LOAD RESUME YES SHRLEVEL CHANGE utility executions, because the utility functions similar to a MASS INSERT when inserting to indexes.

Because DB2 cannot avoid waiting for I/O when reading the clustering index to find the candidate data page, I/O parallelism cannot be performed against the clustering index. An I/O is orders of magnitude slower than CPU processing time. Asynchronous I/O can be scheduled for the last index as long as there is a sufficient number of indexes, 2 or 3, depending on the situation described in the following section.

In general, this enhancement benefits tables with three or more indexes defined. The exceptions are tables defined as MEMBER CLUSTER, tables created with the APPEND YES option, and tables created with the ORGANIZE BY HASH clause. In these cases, indexes are not really used to position the rows in the table, so I/O parallelism is possible with two (secondary) indexes.

I/O parallelism for index update is active in CM. A rebind or bind is not required. However, it is only available for classic partitioned table spaces and universal table spaces (partition-by-growth and range-partitioned). Segmented table spaces are not supported.

Index I/O parallelism likely reduces insert elapsed time and class 2 CPU time. Elapsed time savings are greatest when I/O response times are high. Due to the extra overhead of a sequential prefetch, DBM1 service request block (SRB) time increases, and the total CPU time increases. However, in DB2 10, because the prefetch SRB time is zIIP eligible, the total cost of the CPU time can be reduced. Figure 2-13 is a summary of I/O parallelism for index updates.

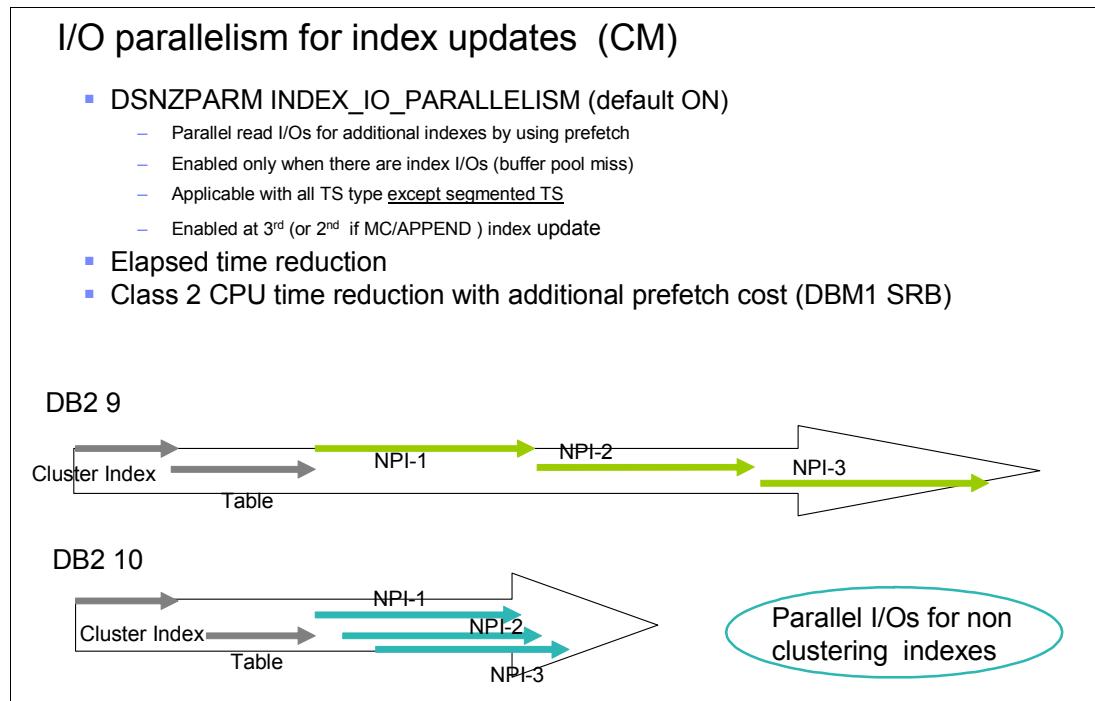


Figure 2-13 I/O parallelism for index updates

I/O parallelism for index updates can also be disabled by setting the new online changeable DSNZPARM parameter INDEX_IO_PARALLELISM to NO. The default is YES. You might want to disable this function if the system has insufficient zIIP capacity to redirect the prefetch engine and if the CPU usage is more important than the response time.

With index I/O parallelism, database I/O suspension is replaced by other read I/O suspension and class 2 CPU time is replaced by SRB time which is zIIP eligible.

The new IFCID 357 and IFCID 358 are available in DB2 10 to trace the start and end of index I/O parallel insert processing. You can use these IFCIDs to monitor for each table insert operation the degree of I/O parallelism, which is the number of synchronous I/O waits DB2 has avoided during the insertions into the indexes for a given table row insert.

In Figure 2-14, we executed 2000 random inserts on 6 indexes, 100% cache miss ratio with 15K RPM HDDs. The result was that elapsed time decreases; class 2 CPU decreases; but the overall CPU increases. However, DBM1 SRB time is zIIP eligible.

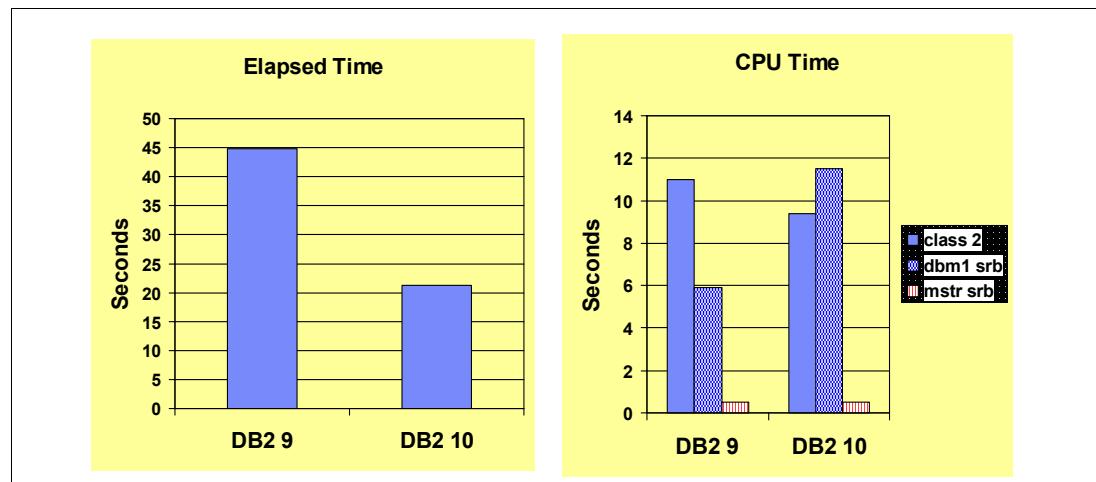


Figure 2-14 Insert index I/O parallelism

2.8 Space search improvement

When DB2 selects a candidate page to insert a new row, it searches for the new key in the cluster index. Let us assume that it is a new key and the key is in the middle of the index (meaning that it is not higher than all other existing keys.) DB2 9 selects the page that contains the next higher key in the cluster index. If that page contains enough space for the new row, the search stops. Otherwise, DB2 must continue the search.

Now suppose that the application tries to insert a series of rows using sequentially higher keys that are all consecutive within the index. Because DB2 9 keeps going back to the same candidate page, it is very likely to run out of space and then have to redo the space search, only to end up inserting the next row in the same page as the previously inserted row. This results in a lot of repetitive and unnecessary space searches.

DB2 10 uses a slightly different algorithm for selecting a candidate page. Instead of using the next higher key in the cluster index, DB2 10 uses the next lower key. This means that if the keys are inserted in ascending order, but into the middle of the index and all at the same insertion point of the index, the candidate page will be the same page as the page where DB2 last found enough free space. Repetitive space searches are avoided.

This behavior helps sequential inserts into the middle of the table based on the clustering index. On the second and subsequent sequential insert, DB2 does not have to repeatedly find the first candidate page as full, which translates directly into CPU and getpage savings because fewer candidate pages need to be searched for sequential insert workloads. This performance improvement is available in CM with no rebind or bind required.

2.9 Log record sequence number spin avoidance for inserts to the same page

DB2 9 in NFM provided a function called *LRSN spin avoidance* that allows for duplicate log record sequence number (LRSN) values for consecutive log records on a given member. Consecutive log records that are written for updates to different pages (for example, a data page and an index page, which is a common scenario) can share the same LRSN value. However, in DB2 9, consecutive log records for the same index or data page must still be unique.

The DB2 member does not need to “spin” consuming CPU under the log latch to wait for the next LRSN increment. This function can avoid significant CPU overhead and log latch contention (LC19) in data sharing environments with heavy logging. See *DB2 9 for z/OS Performance Topics*, SG24-7473 for details. This spin might still be an issue with multi-row INSERT.

DB2 10 further extends LRSN spin avoidance. In DB2 10 NFM, consecutive log records for inserts to the same data page can now have the same LRSN value. If consecutive log records are to the same data page, then DB2 no longer continues to “spin” waiting for the LRSN to increment. The log apply process of recovery has been enhanced to accommodate these duplicate LRSN values.

Duplicate LRSN values for consecutive log records for the same data page set are allowed only for INSERT type log records. DELETE and UPDATE log records still require unique LRSN values.

This enhancement helps applications, such as those applications that use multi-row INSERT where the table or tables that are inserted into have none or only a few indexes, by further reducing CPU utilization.

2.10 Compression on insert

Prior to DB2 10, if you turn on compression for a table space using the CREATE or ALTER TABLESPACE command, DB2 needs to build the compression dictionary. Compression dictionaries are built as part of REORG TABLESPACE or LOAD utility runs. You might not be able to run LOAD or REORG when you decide to turn on compression for a given table space.

With DB2 10 NFM, you can turn on compression with ALTER any time, and the compression dictionary is built when you execute the following statements:

- ▶ INSERT statements
- ▶ MERGE statements
- ▶ LOAD SHRLEVEL CHANGE

Additionally, when you LOAD XML data, a dictionary can be built specifically for the XML table space so that the XML data is compressed in the following circumstances:

- ▶ The table space or partition is defined with COMPRESS YES.
- ▶ The table space or partition has no compression dictionary built yet.
- ▶ The amount of data in the table space is large enough to build the compression dictionary.

There is a 1.2 MB threshold that is determined by reading the RTS statistics in memory. When the threshold is reached, DB2 builds the dictionary asynchronously and issues the DSNU241I message for table space partitioned and DSNU231I for a non-partitioned table space. After the dictionary is built, DB2 inserts the data in compressed format. Figure 2-15 shows an example for a partitioned table space.

```
DSNU241I -DBOB DSNUZLCR - DICTIONARY WITH 4096 755  
ENTRIES HAS BEEN SUCCESSFULLY BUILT FROM 598 ROWS FOR TABLE SPACE  
SABI6.TS6, PARTITION 1
```

Figure 2-15 Message DSNU241I compression dictionary built

If you run the DSN1COMP utility without REORG option, the utility calculates the estimated space savings based on the algorithms that are used for building compression during LOAD, which gets similar compression rates as compress on INSERT.

If you use image copies as input for the UNLOAD utility and you plan on using this automatic compression without REORG or LOAD, you must run the COPY utilities with option SYSTEMPAGES YES, because DB2 builds your compression dictionary on-the-fly. As a result the dictionary pages do not follow the system pages and the UNLOAD utility recognizes that the rows in the table space are compressed but cannot uncompress the rows because it looks only for dictionary pages at the beginning of the table space prior to first data page.

There is no measurable performance impact of using compress on insert. DB2 10 is able to build a compression dictionary automatically after at least 1.2 MB were inserted. In some cases, it might appear that you turn on compression and that the data volume is larger than 1.2 MB but compress on insert did not take place. This situation can occur if DB2 cannot build a usable compression dictionary because of the data contents.



Synergy with z platform

DB2 10 for z/OS takes advantage of the latest improvements in the z platform. DB2 10 increases the synergy with System z hardware and software to provide better performance, more resilience, and better function for an overall improved value.

DB2 benefits from large real memory and faster processors as well as large page frame support. DB2 uses the enhanced features of the storage servers, such as the IBM System Storage® DS8000®. FlashCopy® is used by DB2 utilities, allowing higher levels of availability and ease of operations.

In this chapter, we discuss the following topics:

- ▶ 1 MB page frame support
- ▶ zIIP usage with DB2 10
- ▶ Open and close data sets
- ▶ Disk storage enhancements
- ▶ SMF compression

3.1 1 MB page frame support

The processor uses tables to translate virtual addresses into real memory addresses. The system caches these tables in a translation look-aside buffer (TLB.) Because the size of the translation tables is related to the page frame size, a 1 MB page frame yields significantly smaller translation tables than 4 KB page frames. Reducing the size of these tables improves TLB efficiency, which increases MIPS.

Since DB2 V8 began to remove some of 31-bit virtual storage constraints, and DB2 10 has almost eliminated them, a single DB2 system is now capable of exploiting more memory. However, translation look-aside buffer (TLB) sizes have remained relatively small due to hardware limitations. This results in increased number of TLB misses, which lowers the MIPS.

DB2 10 requires z/OS 1.10 which supports the 1 MB page size on z10 and z196 processors. If the requirements are satisfied, a single TLB entry supports more address translations, which increases TLB hits and raises the MIPS. which also results in better TLB coverage. z/OS supports both 4 KB and 1 MB page sizes.

Large pages are treated as fixed pages and cannot be paged out to auxiliary storage.

To use the large pages, you need to specify the amount of the real storage to be used for large pages by specifying the following keyword in IEASYSxx:

```
LFArea = (xx%| xxxxxM| xxxxxG| xxxxxT)
```

Where xx% indicate the percentage of online storage at IPL time to be used for the large pages, and xxxxxM, xxxxxG, xxxxxT is amount of online storage at IPL time to be used for large pages. The LFAREA is set to zero by default, and you can specify up to 80% of the online storage.

In general, large page is a special purpose feature to improve performance for long running memory intensive applications, not intended for general use because large pages are not pageable.

DB2 10 exploits 1 MB large page frame with buffer pools defined with PGFIX(YES). If LFAREA is specified in IEASYSxx PARMLIB member, DB2 10 uses 1 MB large page frame. We observed 4 to 5% CPU reduction in preliminary measurement.

PGFIX(YES) buffer pools are non-pageable. Over use of PGFIX(YES) buffer pools can cause real storage stress and become the cause for high paging of other application storage.

The current z/OS maintenance for large frames requires z/OS APAR OA31116, OA33702, and OA33529.

3.2 zIIP usage with DB2 10

The special purpose processor zIIP has been made available initially on the IBM System z9® Enterprise Class and z9 Business Class servers which benefit the customer by reducing software charges and freeing the general purpose processors for other work. zIIPs are used to redirect specialized processing elements of DB2. The initial elements were associated with distributed processing, utilities, and complex queries.

DB2 10 adds the following lists of functional redirection to zIIP:

- ▶ More queries make use of query parallelism which in turn introduces additional zIIP eligibility.
- ▶ Portions of the RUNSTATS utility are eligible to be redirected to run on zIIP.
- ▶ Asynchronous I/O for buffer pools which includes dynamic prefetch, list prefetch, sequential prefetch, and deferred write processing, is redirected to run on zIIP.

In this section we discuss zIIP eligibility in the following two situations:

- ▶ RUNSTATS zIIP eligibility
- ▶ Asynchronous I/O zIIP eligibility

3.2.1 RUNSTATS zIIP eligibility

Starting from DB2 10, a portion of the RUNSTATS utility is made eligible for zIIP redirection. Less complex statistics (for example, frequency statistics) will get most of their execution eligible for zIIP. However, more complex statistics requiring that DB2 call to a sort product, such as DFSORT, can benefit from zIIP eligibility of the sort product. For instance, it can reach up to 99.9% for RUNSTATS with no additional parameters and it goes down for more complex statistics such as frequency statistics. Therefore, you might find a varying degree of zIIP eligibility when you execute your RUNSTATS utility workload.

There is no zIIP eligibility for inline statistics.

RUNSTATS measurements

We measured CPU time and elapsed time with various types of RUNSTATS utility jobs executed against different numbers of objects.

We used the following measurement environment:

- ▶ z/OS 1.12
- ▶ DS8300 DASDs
- ▶ z10 4 way and 1 zIIP

We measured RUNSTATS against a table with the following characteristics:

- ▶ Table:
 - 26 columns, 118 byte length
 - 100 million rows
 - 20 partitions
- ▶ Number of indexes: 6

Measurements were done for the following cases with different types and number of indexes:

- ▶ RUNSTATS TABLESPACE
- ▶ RUNSTATS TABLESPACE TABLE
- ▶ RUNSTATS TABLESPACE TABLE INDEX
- ▶ RUNSTATS INDEX PI
- ▶ RUNSTATS INDEX NPI
- ▶ RUNSTATS INDEX 6 INDEXES

Measurement results show that most of the CPU is eligible for zIIP redirection with our test cases where there are no complex statistics to be gathered (Figure 3-1).

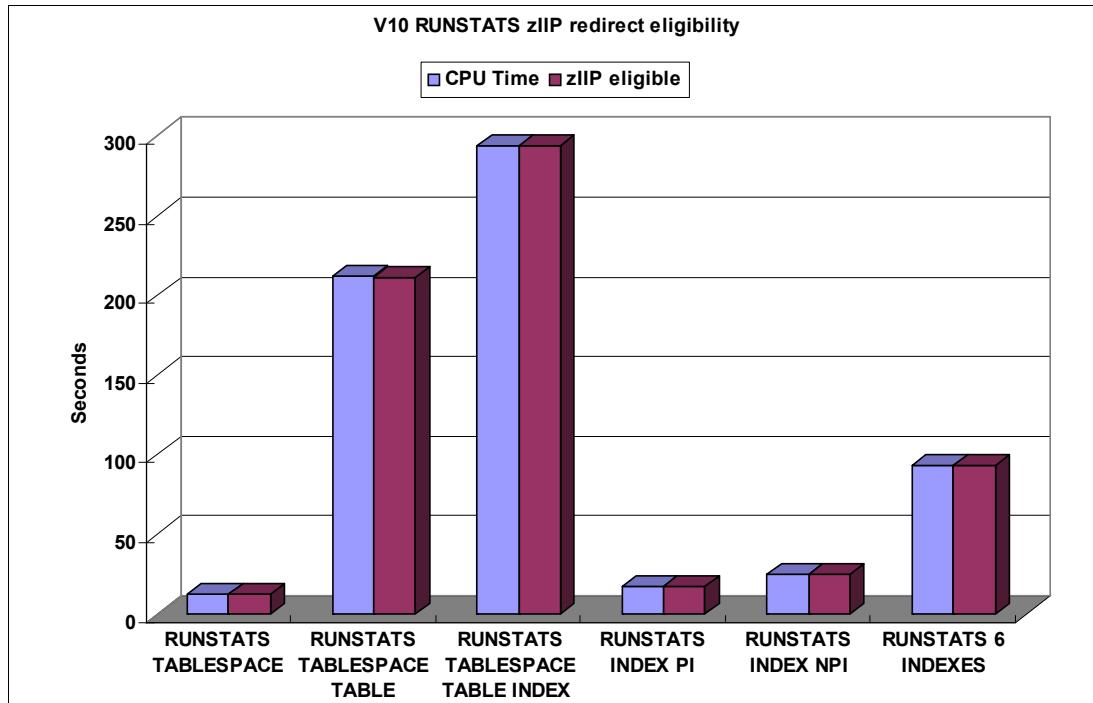


Figure 3-1 RUNSTATS zIIP redirection eligibility

In our environment, measurement was done with only 1 zIIP available. CPU time redirected to zIIP varied between 57% and 98%, with most CPU time still eligible for redirection, and no significant elapsed time increase with CPU times partially redirected to zIIP (Figure 3-2).

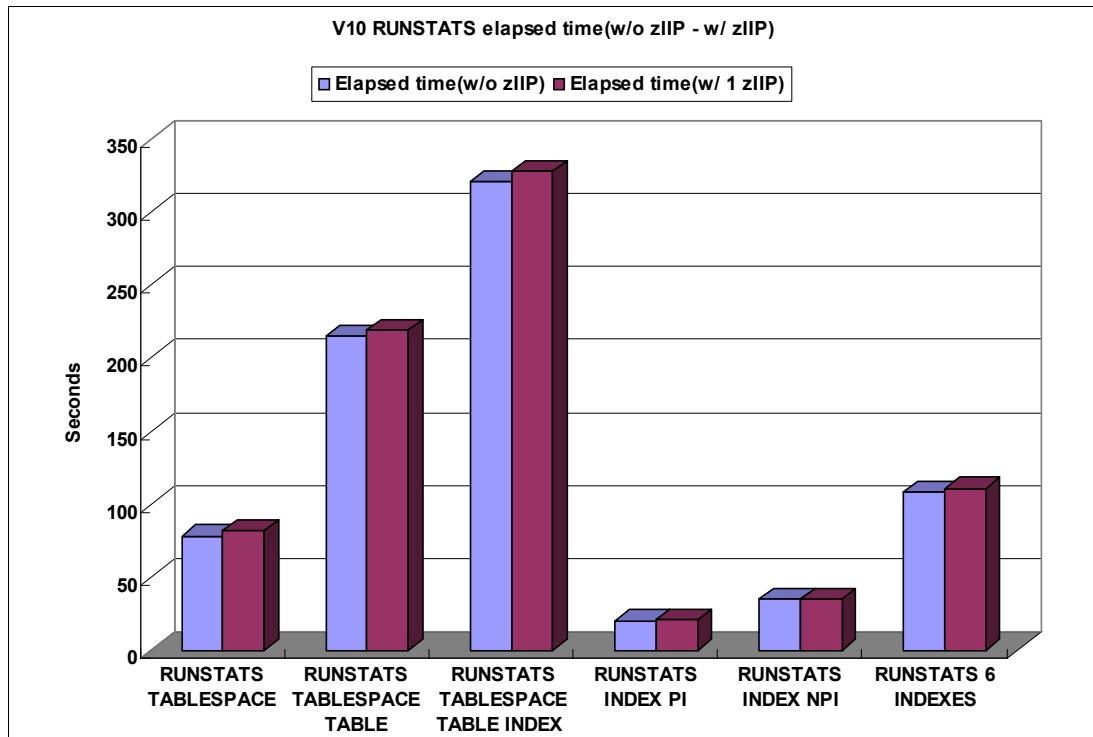


Figure 3-2 RUNSTATS elapsed time with one zIIP

DB2 10 RUNSTATS TABLESAMPLE can deliver large CPU savings when compared to no sampling and compared to traditional row sampling. DB2 10 RUNSTATS TABLESAMPLE elapsed time savings depend on the complexity of the statistics collected and the percentage of data pages being sampled.

RUNSTATS zIIP eligibility reporting

You can report the CPU time routed to a zIIP through DB2 accounting and statistics traces. However, DB2 10 does not provide you with the CPU time that did not run on zIIP but is still eligible for it. Figure 3-3 here, and Figure 3-7 on page 56 show sample output from an OMEGAMON PE report.

MAINPACK : DSNUTIL	CORRNMBR: 'BLANK'	LUW INS: C6A4809D233B			
PRIMAUTH : DB2R5	CONNTYPE: UTILITY	LUW SEQ: 57			
ORIGAUTH : DB2R5	CONNECT : UTILITY				
<hr/>					
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	ELAPSED TIME
-----	-----	-----	-----	-----	-----
ELAPSED TIME	4.840189	4.778304	N/P	LOCK/LATCH(DB2+IRLM)	0.000000
NONNESTED	4.840189	4.778304	N/A	IRLM LOCK+LATCH	0.000000
STORED PROC	0.000000	0.000000	N/A	DB2 LATCH	0.000000
UDF	0.000000	0.000000	N/A	SYNCHRON. I/O	0.057179
TRIGGER	0.000000	0.000000	N/A	DATABASE I/O	0.007112
				LOG WRITE I/O	0.050067
CP CPU TIME	0.021537	0.008329	N/P	OTHER READ I/O	4.053438
AGENT	0.021537	0.008329	N/A	OTHER WRTE I/O	0.000000
NONNESTED	0.021537	0.008329	N/P	SER.TASK SWTCH	0.021068
STORED PRC	0.000000	0.000000	N/A	UPDATE COMMIT	0.011318
UDF	0.000000	0.000000	N/A	OPEN/CLOSE	0.000000
TRIGGER	0.000000	0.000000	N/A	SYSLGRN REC	0.009750
PAR.TASKS	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000
				OTHER SERVICE	0.000000
SECP CPU	0.000000	N/A	N/A	ARC.LOG(QUIES)	0.000000
				LOG READ	0.000000
SE CPU TIME	0.329314	0.329314	N/A	DRAIN LOCK	0.000000

Figure 3-3 DB2 RUNSTATS utility accounting report

Use RMF reports for batch workload monitoring and zIIP capacity planning. As a prerequisite for RMF reporting, you need to perform the following tasks:

- ▶ Define a WLM service classification rule for the batch job or the group of jobs that you want to monitor. For ease of use, make use of RMF report classes in your WLM classification.
- ▶ Configure SMF and RMF to gather the RMF SMF records.
- ▶ Upon successful job completion, create an RMF II workload activity report.

The RMF zIIP reporting is explained in *DB2 9 for z/OS Technical Overview*, SG24-7330.

In the WLM service classification shown in Figure 3-4, we assign the WLM service class BATCHMED to any job that runs in class A with a job name starting with RS. For ease of RMF reporting, we assign a report class of RUNSTATS.

```

Subsystem-Type Xref Notes Options Help
-----
Modify Rules for the Subsystem Type      Row 1 to 8 of
Command ==> _____          Scroll ==> PAGE

Subsystem Type . : JES      Fold qualifier names? Y (Y or N)
Description . . . Batch Jobs

Action codes:   A=After     C=Copy M=Move     I=Insert rule
                B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                More ==>
-----Qualifier-----      -----Class-----
Action    Type      Name      Start      Service      Report
          TC        A         —          BATCHLOW    BATCHDEF
          TN        RS*       —          BATCHLOW    LOWJES2
          —         —         —          BATCHMED   RUNSTATS

```

Figure 3-4 WLM classification for batch job

Upon successful job completion, we created an RMF workload activity report for the RUNSTATS report class using the JCL shown in Example 3-1. Specify the SMF output as MFPPINPUT for input to the RMF post processor.

Example 3-1 JCL RMF workload activity report

```

//RMFPP      EXEC PGM=ERBRMFPP
//MFPPINPUT DD  DISP=SHR,DSN=DB2R5.RS.RMF
//MFPMMSGDS DD  SYSOUT=*
//SYSOUT    DD  SYSOUT=*
//SYSIN    DD  *
      SYSRPTS(WLMGL(RCPER(RUNSTATS)))
      RTOD(0220,0225)
      DINTV(0100)
      SUMMARY(INT,TOT)
      NODELTA
      NOEXITS
      MAXPLEN(225)
      SYSOUT(A)

```

To get all the WLM policy service class and reporting class report information, you can use the following SYSIN control card:

```
SYSRPTS(WLMGL(POLICY,SCPER,SCLASS,RCPER,RCLASS))
```

Figure 3-5 shows the RMF workload activity report for the RUNSTATS report class.

REPORT BY: POLICY=WLMPOL REPORT CLASS=RUNSTATS PERIOD=1

-TRANSACTIONS- TRANS-TIME HHH.MM.SS.TT --DASD I/O-- ---SERVICE--- SERVICE TIME ---APPL %---

		ACTUAL	5.016	SSCHRT	0.2	IOC	63	CPU	0.359	CP	0.01
Avg	0.01	EXECUTION	4.892	RESP	0.4	CPU	10182	SRB	0.000	AAPCP	0.00
MPL	0.01			CONN	0.3	MSO	1173	RCT	0.000	IIPCP	0.00
ENDED	1	QUEUED	123								
END/S	0.00	R/S AFFIN	0	DISC	0.0	SRB	3	IIT	0.002		
#SWAPS	1	INELEGIBLE	0	Q+PEND	0.1	TOT	11421	HST	0.000	AAP	0.00
EXCTD	0	CONVERSION	0	IOSQ	0.0	/SEC	22	AAP	0.000	IIP	0.06
Avg ENC	0.00	STD DEV	0					IIP	0.329		
REM ENC	0.00							ABSRPTN	2347		
MS ENC	0.00							TRX SERV	2347		

GOAL: EXECUTION VELOCITY 20.0% VELOCITY MIGRATION: I/O MGMT 100% INIT MGMT 100%

RESPONSE TIME EX PERF AVG --EXEC USING%-----

SYSTEM	VEL%	INDX	ADRSP	CPU	AAP	IIP	I/O	TOT	
SC63	--N/A--	100	0.2	0.0	0.0	0.0	2.7	16	0.0

EXEC DELAYS % ----- -USING%- --- DELAY % --- %

CRY	CNT	UNK	IDL	CRY	CNT	QUI
0.0	0.0			81	0.0	0.0
				0.0		

Figure 3-5 RMF workload activity report for the RUNSTATS report class

The RMF report shown in Figure 3-5 provides information about zIIP utilization in the SERVICE TIME and the APPL % report blocks. In our example we ran a RUNSTATS utility to collect standard statistics for a table space (all tables and indexes of that table space). In DB2 10, you can expect RUNSTATS to redirect some of its processing to run on zIIP processors.

The RMF SERVICE TIME report block shows a total CPU time of 0.359 seconds (includes CP and zIIP) and a total IIP (zIIP) time of 0.329 seconds, which indicates a RUNSTATS zIIP redirect ratio of about 91% ($0.329 * 100 / 0.359$). If you want to verify whether there was any zIIP eligible time processed on a CP, review the information about IIPCP (IIP processed on CP in percent) that is provided in the RMF APPL % report block.

In our RUNSTATS utility example, the value for IIPCP is zero, indicating that there was no zIIP eligible work processed on a CP. Therefore, we come to the conclusion that there were sufficient zIIP resources available at the time of RUNSTATS utility execution.

To reconfirm that the RUNSTATS utility zIIP eligible time matches the zIIP eligible time shown in the RMF report, we created a DB2 accounting report for the interval in question. The IBM specialty engine time (SE CPU TIME) of the accounting report shown in Figure 3-3 on page 53 matches the zIIP eligible time of the RMF workload activity report in Figure 3-5.

Prefetch and deferred write will be reported against DB2 address space, where APAR PM30468 made changes so zIIP CPU time will be reported against DBM1 address space instead of MSTR address space in the RMF workload activity report reporting class information as shown in Figure 3-6.

See also *DB2 10 for z/OS Technical Overview*, SG24-7892, for details on setting the environment.

REPORT BY: POLICY=OVER			REPORT CLASS=DBOADB1						
			HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS STCHI						
-TRANSACTIONS-			--DASD I/O--						
AVG	1.00	ACTUAL	0	SSCHRT	0.0	IOC	0	CPU	0.208
MPL	1.00	EXECUTION	0	RESP	0.2	CPU	5899	SRB	0.004
ENDED	0	QUEUED	0	CONN	0.1	MSO	60128	RCT	0.000
END/S	0.00	R/S AFFIN	0	DISC	0.0	SRB	124	IIT	0.021
#SWAPS	0	INELIGIBLE	0	Q+PEND	0.1	TOT	66151	HST	0.000
EXCTD	0	CONVERSION	0	IOSQ	0.0	/SEC	37	AAP	0.000
AVG ENC	0.00	STD DEV	0					IIP	0.01
REM ENC	0.00					ABSRPTN	37		
MS ENC	0.00					TRX SERV	37		
GOAL: EXECUTION VELOCITY 60.0%			VELOCITY MIGRATION: I/O MGMT 0.0% INIT MGMT 0.0%						
RESPONSE TIME EX			PERF AVG --EXEC USING%----- EXEC DELAYS % -----						
SYSTEM VEL% INDX ADRSP			CPU AAP IIP I/O TOT						
SC63 --N/A--			0.0 0.0 2.0 0.0 0.0 0.0 0.0 0.0						

Figure 3-6 RMF workload activity report for the DBM1 address space report class

3.2.2 Asynchronous I/O zIIP eligibility

With DB2 10, asynchronous I/O executed by buffer pool prefetch engines and deferred write engines is 100% zIIP eligible. Castout engines work is not eligible for zIIP.

Buffer pool prefetch includes dynamic prefetch, list prefetch, and sequential prefetch activities. Buffer pool prefetch activities are asynchronously initiated by the database manager address space (DBM1) and are executed in a dependent enclave. Because asynchronous services buffer pool prefetch activities are not accounted to the DB2 client, they show up in the DB2 statistics report, where you can see CPU times accounted to zIIP in PREEMPT IIP SRB (Figure 3-7).

CPU TIMES	TCB TIME	PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME	PREEMPT IIP SRB
<hr/>					
SYSTEM SERVICES ADDRESS SPACE	2.565794	0.825791	0.310654	3.702240	N/A
DATABASE SERVICES ADDRESS SPACE	0.187984	0.094044	0.003409	0.285437	0.486775
IRLM	0.000002	0.000000	0.065226	0.065228	N/A
DDF ADDRESS SPACE	0.005344	0.000000	0.000105	0.005448	0.000000
TOTAL	2.759124	0.919835	0.379394	4.058353	0.486775

Figure 3-7 OMEGAMON PE statistics trace sample

Our measurement showed that redirection to zIIP was significant with index compression and insert processing with index I/O parallelism.

In this section we provide examples of zIIP measurements using OMEGAMON PE and RMF™.

DB2 10 zIIP eligible buffer pool prefetch activities are asynchronously initiated by the database manager address space (DBM1) and are executed in a dependent enclave. With APAR PM30468 (PTF UK64423), prefetch and deferred write CPU, when running on a zIIP processor, is reported by WLM under the DBM1 address space, not under the MSTR.

If you want to use the IBM Resource Measurement Facility™ (RMF™) for zIIP reporting and monitoring, you need to collect the SMF RMF records as described in *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6645. For more references about the use of RMF reports, see the IBM website, z/OS 1.12.0 elements and featured PDF files, section RMF, at <http://www.ibm.com/systems/z/os/zos/bkserv/r12pdf/#rmf>.

Using OMEGAMON PE

To evaluate the DBM1 zIIP CPU utilization, proceed as follows:

1. Switch SMF using the command /I SMF. This step helps to reduce the scope of the SMF data to analyze.
2. Issue a MODIFY TRACE command to produce a new statistics record before starting your testing.
3. As the *single* user of the DB2 subsystem, perform your tests several times to produce a valid average.
4. Issue again a MODIFY TRACE command to cut another statistics record.
5. (Optional) Issue an SMF switch command to have SMF data immediately available for analysis (/I SMF)

The previous steps can be executed safely in a test environment. Check the potential impacts on the SMF collection process with your z/OS System administrator before attempting these steps in a heavy loaded or a production system.

The sequence of steps indicating the time at which each event occurred is documented in Figure 3-8. This chart can be used in order to better understand the process described in the following lines of this section.

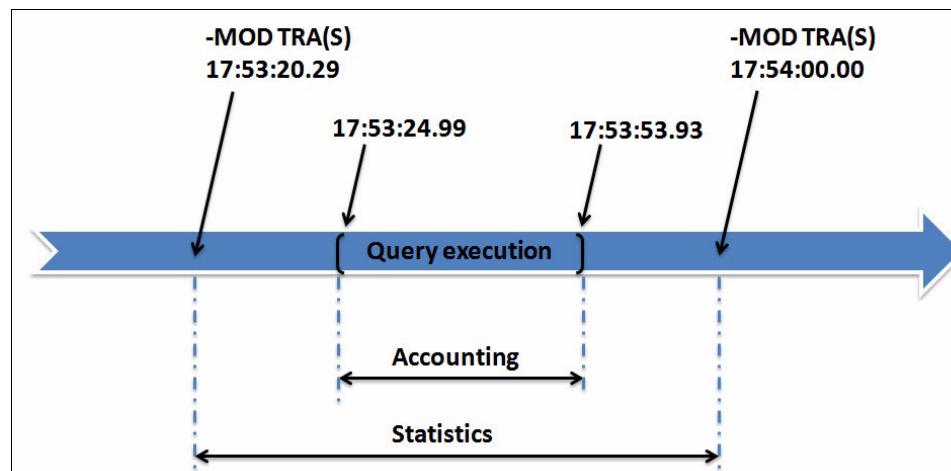


Figure 3-8 DB2 accounting and DB2 statistics

At this stage, it is important to understand that the statistics trace reports the resource utilization for all the DB2 activity occurred in each period. Statistics trace is a system-wide trace and must be apportioned for charge back accounting. Use the information that the statistics trace provides to plan DB2 capacity, or to tune the entire set of active DB2 programs.

The STATIME subsystem parameter specifies the time interval, in minutes, between statistics collections. Statistics records are to be written at approximately the end of this interval. In DB2 10, the STATIME subsystem parameter applies only to IFCIDs 0105, 0106, 0199, and 0365. IFCIDs 0001, 0002, 0202, 0217, 0225, and 0230 are no longer controlled by STATIME, and the corresponding trace records are written at fixed, one-minute intervals. We used a 1 minute interval for the tests described in this section.

In order to report the DB2 address spaces zIIP CPU utilization due to a specific process or query by using the DB2 statistics traces, you need to execute ONLY that specific process or query during the studied interval.

As an example, consider the very simple query in Example 3-2. This query returns no result set, no rows qualify. This was done intentionally in order to avoid data transfer interferences in the observed measurements.

Example 3-2 Simple query example, table space scan

```
SELECT *
FROM
TABLE1
WHERE T_DTT = 'OK'
WITH UR ;
```

The target table contains more than 12 millions rows and no index defined. As a result, this sample query is solved with a table space scan access path. We used DB2 Data Studio for exploring the access path. Figure 3-9 indicates that the access path involved a sequential prefetch operation.

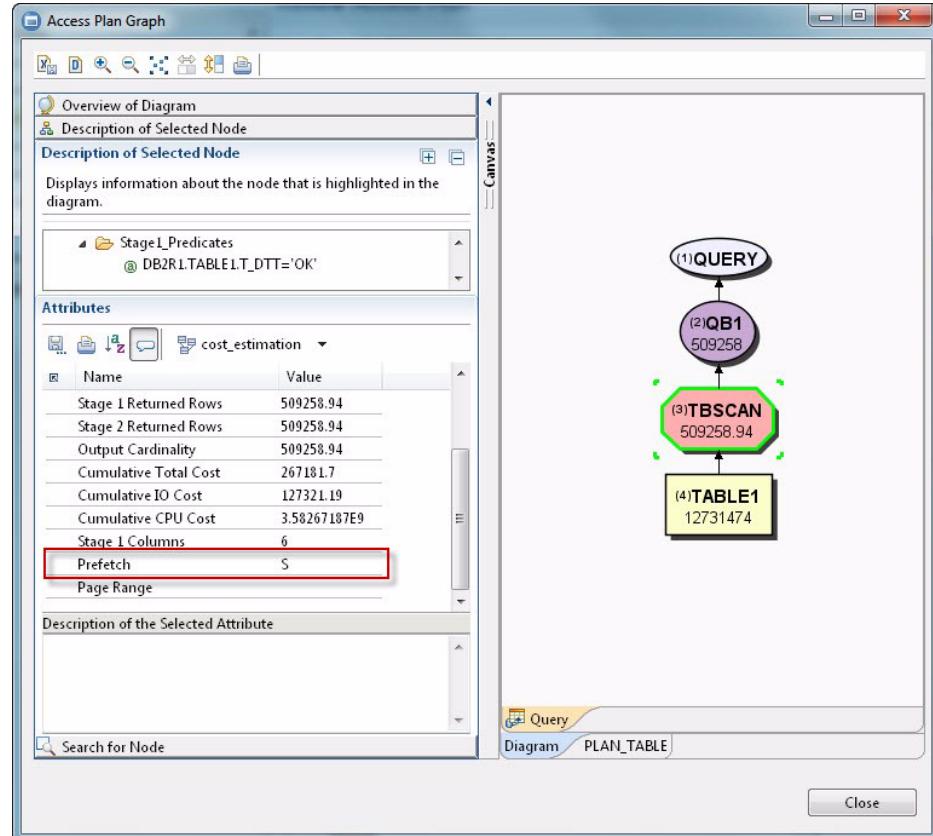


Figure 3-9 DB2 Data Studio, access path indicating sequential prefetch

Example 3-3 shows how we executed the MODIFY TRACE command using a JCL.

Example 3-3 Modify Trace JCL example

```
/*-----  
/* -MODIFY TRACE --> WRITES A STATISTICS RECORD  
/*-----  
//MODSTAT EXEC PGM=IKJEFT01,DYNAMNBR=20  
//STEPLIB DD DSN=DBOA.SDSNLOAD,DISP=SHR  
//SYSTSPRT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=*  
//SYSUDUMP DD SYSOUT=*  
//SYSTSIN DD *  
    DSN SYSTEM(DBOA)  
    -DIS TRACE(*)  
    -MOD TRA(S) CLASS(1,3,4,5,6) TNO(01)  
*/
```

Example 3-4 shows an example of Modify Trace command execution results.

Example 3-4 Modify Trace execution example

```
READY  
    DSN SYSTEM(DBOA)  
DSN  
    -DIS TRACE(*)  
DSNW127I -DBOA CURRENT TRACE ACTIVITY IS -  
TNO TYPE CLASS DEST QUAL IFCID  
01 STAT 01,03,04,05, SMF NO  
01      06  
02 ACCTG *           SMF NO  
*****END OF DISPLAY TRACE SUMMARY DATA*****  
DSN9022I -DBOA DSNWVCM1 '-DIS TRACE' NORMAL COMPLETION  
DSN  
    -MOD TRA(S) CLASS(1,3,4,5,6) TNO(01)  
DSNW130I -DBOA S TRACE STARTED, ASSIGNED TRACE NUMBER 01  
DSN9022I -DBOA DSNWVCM1 '-MOD TRA' NORMAL COMPLETION  
DSN  
*/  
DSN  
END
```

The accounting results due to these query executions were reported using OMEGAMON PE using the JCL in Example 3-5.

Example 3-5 OMEGAMON PE JCL for reporting DB2 prefetch

```
//PE      EXEC PGM=FPECMAN  
//STEPLIB DD DISP=SHR,DSN=OMEGASYS.DBOA.BASE.RKANMOD  
//INPUTDD DD DISP=SHR,DSN=SMFDATA.ALLRECS.G8858V00  
//JOBSUMDD DD SYSOUT=*  
//SYSOUT   DD SYSOUT=*  
//ACRPTDD DD SYSOUT=*  
//UTTRCDD1 DD SYSOUT=*  
//SYSIN   DD *  
GLOBAL  
    TIMEZONE (+ 05:00)  
ACCOUNTING  
    TRACE  
    LAYOUT(LONG)
```

```

INCLUDE(SUBSYSTEM(DB0A))
INCLUDE(PRIMAUTH(DB2R1))
STATISTICS
TRACE
EXEC
/*

```

Example 3-6 shows a portion of an OMEGAMON PE Accounting Trace long report, indicating that the query was executed using DSNESPCS, that is, the SPUFI package with isolation Cursor Stability. The BP0 buffer pool activity section indicates the execution of 19899 sequential prefetch requests and 636708 getpages.

The SEQ. PREFETCH REQS field indicates the number of SEQUENTIAL PREFETCH requests, and this value is incremented for each PREFETCH request. Each request can result in an I/O read. For SQL statements, depending on the buffer pool size, a request does not result in an I/O if all the requested pages are already in the buffer pool. See the Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Version 5.1 documentation for details on the report and fields.

Example 3-6 OMEGAMON PE Accounting: sequential prefetch

---- IDENTIFICATION -----			
ACCT TSTAMP: 03/08/11 17:53:53.93	PLANNNAME: DSNESPCS	WLM SCL: 'BLANK'	CICS NET: N/A
BEGIN TIME : 03/08/11 17:53:24.99	PROD TYP: N/P		CICS LUN: N/A
END TIME : 03/08/11 17:53:53.93	PROD VER: N/P	LUW NET: USIBMSC	CICS INS: N/A
REQUESTER : DB0A	CORRNNAME: DB2R1	LUW LUN: SCPDB0A	
MAINPACK : DSNESM68	CORRNMBR: 'BLANK'	LUW INS: C7710C394793	ENDUSER : 'BLANK'
PRIMAUTH : DB2R1	CONNTYPE: TSO	LUW SEQ: 3	TRANSACT: 'BLANK'
ORIGAUTH : DB2R1	CONNECT : TSO		WSNAME : 'BLANK'
BP0 BPOOL ACTIVITY	TOTAL		
BPOOL HIT RATIO (%)	0		
GETPAGES	636708		
BUFFER UPDATES	0		
SYNCHRONOUS WRITE	0		
SYNCHRONOUS READ	3		
SEQ. PREFETCH REQS	19899		
LIST PREFETCH REQS	0		
DYN. PREFETCH REQS	0		
PAGES READ ASYNCHR.	632871		

Example 3-7 shows a portion of an OMEGAMON PE report listing the application CPU time distribution.

Example 3-7 OMEGAMON PE accounting: special engine CPU time

---- IDENTIFICATION -----							
ACCT TSTAMP: 03/08/11 17:53:53.93	PLANNNAME: DSNESPCS	WLM SCL: 'BLANK'	CICS NET: N/A				
BEGIN TIME : 03/08/11 17:53:24.99	PROD TYP: N/P		CICS LUN: N/A				
END TIME : 03/08/11 17:53:53.93	PROD VER: N/P	LUW NET: USIBMSC	CICS INS: N/A				
REQUESTER : DB0A	CORRNNAME: DB2R1	LUW LUN: SCPDB0A					
MAINPACK : DSNESM68	CORRNMBR: 'BLANK'	LUW INS: C7710C394793	ENDUSER : 'BLANK'				
PRIMAUTH : DB2R1	CONNTYPE: TSO	LUW SEQ: 3	TRANSACT: 'BLANK'				
ORIGAUTH : DB2R1	CONNECT : TSO		WSNAME : 'BLANK'				
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	IFI (CL.5)	CLASS 3 SUSPENSIONS	ELAPSED TIME	EVENTS	HIGHLIGHTS
ELAPSED TIME	28.937422	27.632313	N/P	LOCK/LATCH(DB2+IRLM)	0.000078	48	THREAD TYPE : ALLIED
NONNESTED	28.937422	27.632313	N/A	IRLM LOCK+LATCH	0.000000	0	TERM.CONDITION: NORMAL
STORED PROC	0.000000	0.000000	N/A	DB2 LATCH	0.000078	48	INVOKE REASON : DEALLOC
UDF	0.000000	0.000000	N/A	SYNCHRON. I/O	0.001102	3	COMMITS :
TRIGGER	0.000000	0.000000	N/A	DATABASE I/O	0.001102	3	ROLLBACK :
				LOG WRITE I/O	0.000000	0	SVPT REQUESTS :
CP CPU TIME	4.368565	4.359003	N/P	OTHER READ I/O	22.878914	19795	SVPT RELEASE :
AGENT	4.368565	4.359003	N/A	OTHER WRTE I/O	0.000000	0	SVPT ROLLBACK :
NONNESTED	4.368565	4.359003	N/P	SER.TASK SWTCH	0.000037	1	INCREM.BINDS :
STORED PRC	0.000000	0.000000	N/A	UPDATE COMMIT	0.000037	1	UPDATE/COMMIT : 0.00
UDF	0.000000	0.000000	N/A	OPEN/CLOSE	0.000000	0	SYNCH I/O AVG.: 0.000367
TRIGGER	0.000000	0.000000	N/A	SYSLGRNG REC	0.000000	0	PROGRAMS :
PAR.TASKS	0.000000	0.000000	N/A	EXT/DEL/DEF	0.000000	0	MAX CASCADE :
				OTHER SERVICE	0.000000	0	PARALLELISM : NO

SECP CPU	0.000000	N/A	N/A	ARC.LOG(QUIES)	0.000000	0
				LOG READ	0.000000	0
SE CPU TIME	0.000000	0.000000	N/A	DRAIN LOCK	0.000000	0
NONNESTED	0.000000	0.000000	N/A	CLAIM RELEASE	0.000000	0
STORED PROC	0.000000	0.000000	N/A	PAGE LATCH	0.000000	0
UDF	0.000000	0.000000	N/A	NOTIFY MSGS	0.000000	0
TRIGGER	0.000000	0.000000	N/A	GLOBAL CONTENTION	0.000000	0
				COMMIT PH1 WRITE I/O	0.000000	0
PAR.TASKS	0.000000	0.000000	N/A	ASYNCH CF REQUESTS	0.000000	0
				TCP/IP LOB XML	0.000000	0

In this example, the field SE CPU TIME represents the sum of several accumulated CPU times consumed while running on an IBM specialty engine in all environments. These times are consumed in the following situations:

- ▶ Running stored procedure requests and triggers on the main application execution unit
- ▶ Satisfying stored procedure requests processed in a DB2 stored procedure or WLM address space; SQLP times are included in this time if the SQLP was called on a nested task and was not invoked by the main application execution unit
- ▶ Satisfying UDF requests processed in a DB2 stored procedure or WLM address space
- ▶ Running triggers on a nested task
- ▶ Running parallel tasks in an application which contains the accumulated CPU time used to satisfy UDF requests

Attention: DB2 10 redirects prefetch CPU time to zIIP specialty engines. CPU time for asynchronous work, such as prefetch, is shown in statistics data, including use of specialty engines.

The zIIP prefetch time is reported in the statistics traces. Example 3-8 illustrates a portion of the OMEGAMON PE Statistics Trace - Short report showing the statistics trace collected for the period indicated in Figure 3-8 between **-MOD TRA(S)** commands. The prefetch zIIP CPU time is accounted in the DATABASE SERVICES ADDRESS SPACE under the PREEMPT IIP SRB column. The field DATABASE SERVICES ADDRESS SPACE - PREEMPT IIP SRB of this report shows the preemptable SRB time for the database services address space consumed on an zIIP processor.

Example 3-8 OMEGAMON PE statistics trace

LOCATION: DBOA	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1) STATISTICS TRACE - SHORT				PAGE: 1-177
GROUP: N/P					REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P					TO: NOT SPECIFIED
SUBSYSTEM: DBOA					ACTUAL FROM: 03/08/11 17:22:00.00
DB2 VERSION: V10					
---- HIGHLIGHTS -----					
BEGIN RECORD: 03/08/11 17:53:20.29	TOTAL THREADS	:	1	AUTH SUCC.W/OUT CATALOG:	1
END RECORD : 03/08/11 17:54:00.00	TOTAL COMMITS	:	4	BUFF.UPDT/PAGES WRITTEN:	N/C
ELAPSED TIME: 39.717998	INCREMENTAL BINDS:	0		PAGES WRITTEN/WRITE I/O:	N/C
					TOTAL API :
					0
CPU TIMES					
	TCB TIME		PREEMPT SRB	NONPREEMPT SRB	TOTAL TIME
-----	-----	-----	-----	-----	-----
SYSTEM SERVICES ADDRESS SPACE	0.002499	0.000206	0.001055	0.003760	N/A
DATABASE SERVICES ADDRESS SPACE	0.000281	3.558289	0.000112	3.558681	0.052862
IRLM	0.000000	0.000000	0.041133	0.041133	N/A
DDF ADDRESS SPACE	0.003684	0.000017	0.000079	0.003781	0.000000
TOTAL	0.006464	3.558513	0.042379	3.607355	0.052862
...					
BPO	GENERAL	QUANTITY	BP32K	GENERAL	QUANTITY
BPOOL HIT RATIO (%)	0.60	BPOOL HIT RATIO (%)	N/C	BPOOL HIT RATIO (%)	N/C
GETPAGES-SEQ&RANDOM	636712	GETPAGES-SEQ&RANDOM	0	GETPAGES-SEQ&RANDOM	2
GETPAGES-SEQ.ONLY	636574	GETPAGES-SEQ.ONLY	0	GETPAGES-SEQ.ONLY	0
SYNC.READ-SEQ&RANDOM	4	SYNC.READ-SEQ&RANDOM	0	SYNC.READ-SEQ&RANDOM	0
SYNC.READ-SEQ.ONLY	1	SYNC.READ-SEQ.ONLY	0	SYNC.READ-SEQ.ONLY	0
SEQ.PREFETCH REQ	19899	SEQ.PREFETCH REQ	0	SEQ.PREFETCH REQ	0
SEQ.PREFETCH READS	19813	SEQ.PREFETCH READS	0	SEQ.PREFETCH READS	0

PAGES READ-SEQ.PREF.	632871	PAGES READ-SEQ.PREF.	0	PAGES READ-SEQ.PREF.	0	PAGES READ-SEQ.PREF.	0
LST.PREFETCH REQUEST	0	LST.PREFETCH REQUEST	0	LST.PREFETCH REQUEST	0	LST.PREFETCH REQUEST	0
...							
TOTAL GENERAL	QUANTITY						
BPOOL HIT RATIO (%)	0.60						
GETPAGES-SEQ&RANDOM	636714						
GETPAGES-SEQ.ONLY	636574						
SYNC.READ-SEQ&RANDOM	4						
SYNC.READ-SEQ.ONLY	1						
SEQ.PREFETCH REQ	19899						
SEQ.PREFETCH READS	19813						
PAGES READ-SEQ.PREF.	632871						
LST.PREFETCH REQUEST	0						

Comparing the accounting and statistics reports you can determinate that all the sequential prefetch request come from our test query; the numbers of sequential prefetch request and getpages match in both reports.

Using RMF

With APAR PM30468 (PTF UK64423), prefetch and deferred write CPU, when running on a zIIP processor, is reported by WLM under the DBM1 address space, not under the MSTR address space. In our test environment we classify each one of the DB2 subsystem address spaces in a different report class, as shown in Example 3-9. A WLM report class aggregates a set of work for reporting purposes. You can use report classes to analyze the performances of individual workloads running in the same or different service classes. Work is classified into report classes using the same classification rules that are used for classification into service classes. A good way to contrast report classes to service classes is that report classes are used for monitoring work, while service classes primarily need to be used for managing work.

Example 3-9 WLM classification of DB2 address spaces into reporting classes

Subsystem-Type	Xref	Notes	Options	Help

Modify Rules for the Subsystem Type Row 3 to 13 of 24				
Command ==> _____ Scroll ==> CSR				
Subsystem Type . . : STC Fold qualifier names? Y (Y or N)				
Description . . . Started Tasks				
Action codes: A=After C=Copy M=Move I=Insert rule B=Before D=Delete row R=Repeat IS=Insert Sub-rule More ==>				
-----Qualifier----- -----Class-----				
Action	Type	Name	Start	Service Report
				DEFAULTS: STC RSYSDFLT
____	1	TN	DBOABDM1 ____	STCHI DBOABDM1
____	1	TN	DBOAMSTR ____	STCHI DBOAMSTR
____	1	TN	DBOADIST ____	STCHI DBOADIST
____	1	TN	DBOAIRLM ____	STCHI DBOAIRLM
____	1	TN	DBOAADMT ____	STCHI DBOAADMT

Example 3-10 shows an RMF post processor JCL for monitoring reporting classes. To facilitate the exhibit of the zIIP CPU utilization we changed the RMF interval to 1 minute. See also 3.2, “zIIP usage with DB2 10” on page 50.

Example 3-10 RMF post processor syntax example

```
//RMFPP EXEC PGM=ERBRMFP
//MFPINPUT DD DISP=SHR,DSN=SMFDATA.ALLRECS.G8858V00
//MFPMSGDS DD SYSOUT=*
//SYSOUT DD SYSOUT=*
```

```

//SYSIN DD *
  SYSRPTS(WLMGL(RCPER(DBOABDM1,DBOAMSTR,DBOADIST,DBOAIRLM)))
  SUMMARY(INT,TOT)
  NODELTA
  NOEXITS
  MAXPLEN(225)
  SYSOUT(A)
/*

```

Example 3-11 shows an RMF workload activity report showing information for the report class=DB0ABDM1; it is work executed by the DB0ABDM1 address space according to the workload classification shown in Example 3-9. The field IIP under the service time section indicates the CPU time spend in a zIIP engine. Compare this report with the OMEGAMON PE Statistics trace, section DATABASE SERVICES ADDRESS SPACE - PREEMPT IIP SRB in Example 3-8; the zIIP CPU time is the same.

Example 3-11 RMF Workload Activity report

WORKLOAD ACTIVITY												PAGE 1
z/OS V1R12		SYSPLEX SANDBOX		DATE 03/08/2011		INTERVAL 01.00.000		MODE = GOAL				
				RPT VERSION V1R12 RMF		TIME 17.54.00						
POLICY ACTIVATION DATE/TIME 03/08/2011 17.40.19												
----- REPORT CLASS PERIODS												
REPORT BY: POLICY=OVER				REPORT CLASS= DBOABDM1				PERIOD=1				
				HOMOGENEOUS: GOAL DERIVED FROM SERVICE CLASS STCHI								
-TRANSACTIONS- TRANS-TIME HHH.MM.SS.TT --DASD I/O-- ---SERVICE--- SERVICE TIME ---APPL %--- --PROMOTED-- ----STORAGE---- AVG 1.00 ACTUAL 0 SSCHRT 0.0 IOC 0 CPU 3.614 CP 6.30 BLK 0.000 AVG 67020.23 MPL 1.00 EXECUTION 0 RESP 0.0 CPU 102525 SRB 0.000 AAPCP 0.00 ENQ 0.000 TOTAL 67018.00 ENDED 0 QUEUED 0 CONN 0.0 MSO 1653K RCT 0.000 IIPCP 0.00 CRM 0.000 SHARED 1.00 END/S 0.00 R/S AFFIN 0 DISC 0.0 SRB 4 IIT 0.221 LCK 0.000 #SWAPS 0 INELIGIBLE 0 Q+PEND 0.0 TOT 1756K HST 0.000 AAP 0.00 -PAGE-IN RATES- EXCTD 0 CONVERSION 0 /SEC 29262 AAP 0.000 IIP 0.09 SINGLE 0.0 AVG ENC 0.00 STD DEV 0 ABSRPTN 29K BLOCK 0.0 REM ENC 0.00 TRX SERV 29K SHARED 0.0 MS ENC 0.00 HSP 0.0												

For zSeries®, see *Effective zSeries Performance Monitoring, Using Resource Measurement Facility*, SG24-6645 for more details about RMF and the RMF postprocessor.

3.3 Open and close data sets

DB2 start up or shut down times can be lengthy, especially when DB2 for z/OS has to go through data set allocation processing for thousands of data sets. z/OS 1.12 delivers functions that allow for significant data set allocation elapsed time reductions. These improvements can result in significant DB2 start up or shut down time improvements.

DB2 10 (or DB2 V8 and 9 with APARs) exploits z/OS 1.12 new allocation functions to improve the performance of allocation, deallocation, open, and close of the data sets in DB2 page sets. These functions significantly improved the performance when opening a large number of data sets concurrently, especially for DB2 users with a high value of DSMAX. Significant reduction in elapsed time has been observed by DB2 performance tests with a DSMAX of 100,000.

The new interface for open and close data sets can be enabled by one of the following actions:

- ▶ Update the ALLOCxx PARMLIB member to set the SYSTEM MEMDSENQMGMT value to ENABLE.
- ▶ Issue the system command:

```
SETALLOC SYSTEM,MEMDSENQMGMT=ENABLE
```

Updating the ALLOCxx PARMLIB is best, because it remains effective across IPLs. If the SETALLOC command is used to enable SYSTEM MEMDSENQMGMT, a DB2 restart is required to make the change effective.

We used the following measurement environment:

- ▶ z/OS 1.11 and 1.12
- ▶ z10
- ▶ DB2 9 with 20 parallel jobs
- ▶ IBM DFSMS/MVS™ Enhanced Catalog Sharing (ECS) enabled

Figure 3-10 shows the measurement results for this enhancement using 20 concurrent jobs to open 100,000 DB2 data sets (on the left) with DSMAX set to 100,000. Figure 3-10 illustrates the number of opened data sets (on the left) and consumed CPU time (on the right) measured (in seconds) every minute. Results were significant: we observed 3 times shorter elapsed time and 6 times shorter DB2 DBM1 CPU time closing 100,000 DB2 data sets with this z/OS 1.12 enhancement.

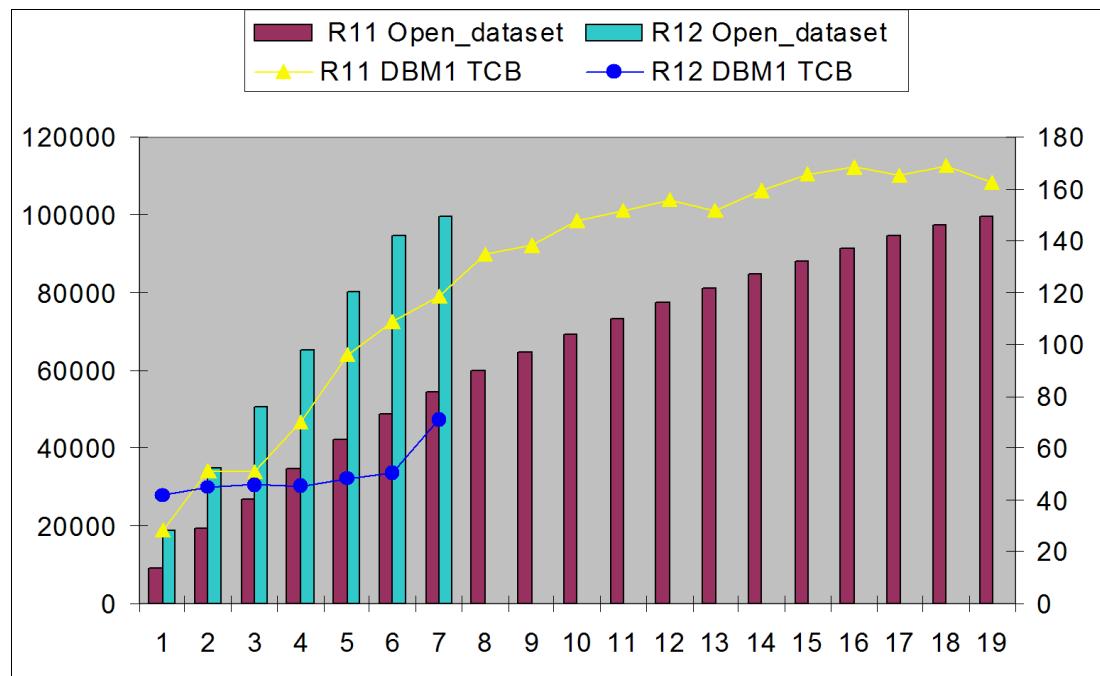


Figure 3-10 Open 100,000 data sets using 20 concurrent jobs

An additional measurement was done in the same environment using 20 concurrent jobs opening 20,000 data sets with DSMAX set to 80,000 and 80,000 data sets being already opened. In this case, where close and open is performed, we observed 4 times shorter elapsed time and CPU time improvement with this z/OS 1.12.

The same benefits are available to DB2 V8 and DB2 9, if you are using z/OS 1.12 or later, with APARs PM00068, PM17542, and PM18557.

3.4 Disk storage enhancements

The IBM System Storage DS8000 series is the flagship disk storage platform within the IBM System Storage product portfolio. Introduced in October 2010, the new DS8800 (IBM 2107 Model 951) represents the latest in this series of high-performance, high-capacity, flexible, and resilient disk storage systems. The most visible change from the DS8300 and DS8700 models is the high density storage enclosure and frame design. The DS8800 provides storage enclosure support for 24 small form factor (SFF), 2.5", 6 Gbps (gigabits per second) serial-attached SCSI (SAS) drives in 2U (rack units) of rack space. This new compact design enables higher scalability, significant footprint reduction, and greater energy savings as compared to previous enclosures that only supported 32 drives in 3.5U of rack space. The most notable changes that enhance performance of the DS8800 include these:

- ▶ 8 Gbps PCIe¹ Host Adapter (HA)
- ▶ 8 Gbps PCIe RAID Device Adapters (DA)

Both the DS8800 and the DS8700 utilize the next generation IBM POWER6® processor within their Central Electronics Complex (CEC) and have replaced the RIO-G I/O enclosure, used on the DS8300, with the Peripheral Component Interconnect Express (PCIe) I/O enclosure. Additionally, the POWER6 CEC in the DS8800 is based on the ultra-high frequency, dual-core IBM POWER6+™ processor technology, which at 5.0 GHz is one of the industry leaders in performance, scalability, and modularity.

A single frame of the DS8300 holds up to 128 drives. The same for DS8700. A single frame of the DS8800 holds up to 240 drives. A DS8800 with 3 frames contains approximately the same number of disks as a fully configured DS8300 or DS8700.

Higher CPU capacity requires greater I/O bandwidth and efficiency. High Performance FICON (zHPF) enhances the IBM z/Architecture® and the FICON interface architecture to provide greater I/O efficiency. zHPF is a data transfer protocol that is optionally employed for accessing data from an IBM DS8000 storage subsystem.

Both the DS8800 and the zHPF provide great improvements when used with DB2 for z/OS. See *DB2 10 for z/OS Technical Overview*, SG24-7892, for additional information.

Measurements were done comparing DS8800 with DS8300 with zHPF and FICON showing how DB2 functions can benefit from the I/O improvements. In this section we discuss the following measurements:

- ▶ Prefetch improvement through disk enhancement
- ▶ DB2 logging and insert with disk enhancements
- ▶ Utilities and storage enhancement

3.4.1 Prefetch improvement through disk enhancement

A set of measurements were done focusing on prefetch operations performed by DB2 for z/OS. DB2 supports three types of prefetch: sequential prefetch, dynamic prefetch, and list prefetch. In most cases, these prefetch types read more than 64 KB bytes and therefore were not eligible for zHPF prior to the z196. The z196 makes sequential prefetch and dynamic prefetch both zHPF-eligible.

¹ PCI Express (Peripheral Component Interconnect Express), officially abbreviated as PCIe, is a computer expansion card standard designed to replace the older PCI bus standards.

Sequential prefetch reading 4 KB pages from cache

The first measurements were done for sequential prefetch using 4 KB pages on a DS8300 and a DS8800 attached to a z196 processor. The z196 enables DB2 prefetch I/Os to become zHPF eligible. The buffer pool is defined large enough to enable DB2 to read 256 KB per I/O while reading from disk cache. See Figure 3-11.

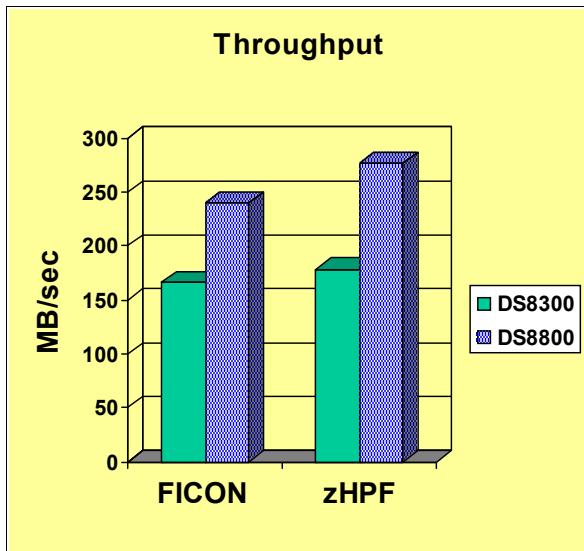


Figure 3-11 FICON versus zHPF for 4 KB page sequential prefetch

The results show that using FICON, the DS8800 channel throughput is 44% higher than the DS8300. Using zHPF, the DS8800 channel throughput is 55% higher than the DS8300. If we compare the DS8800 with zHPF to the DS8300 with FICON, the channel throughput increases 66%.

Dynamic prefetch reading 4 KB pages from cache

The second set of measurements was done for dynamic prefetch using 4 KB pages on a DS8300 and a DS8800 attached to a z196 processor. The z196 processor enables DB2 prefetch I/Os to become zHPF eligible. The measurement conditions are similar to the sequential prefetch case where pages are read from the disk cache. See Figure 3-12.

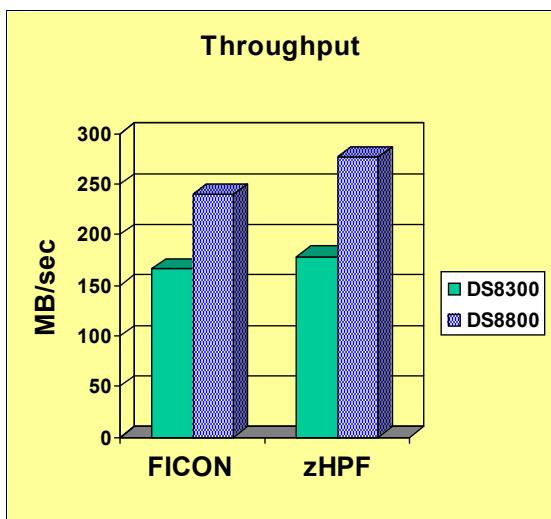


Figure 3-12 FICON versus zHPF for 4 KB page dynamic prefetch

The results show that using FICON, the DS8800 channel throughput is 33% higher than the DS8300. Using zHPF, the DS8800 channel throughput is 38% higher than the DS8300. And together, comparing the DS8800 with zHPF to DS8300 with FICON, channel throughput increases 58%.

Sequential prefetch reading 4 KB pages from disk

The third set of measurements was done with prefetch reading data from disk, not retrieving the data from the disk cache. See Figure 3-13.

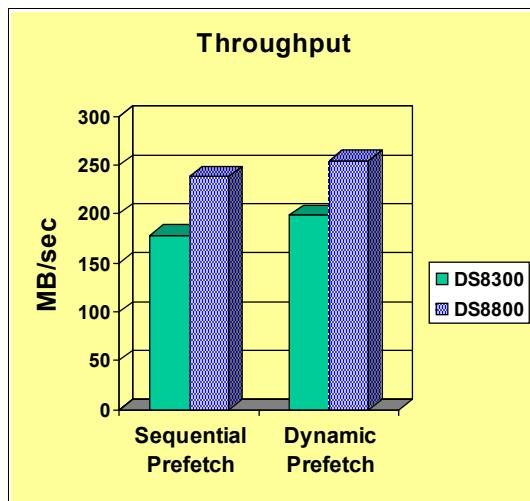


Figure 3-13 The 4 KB page prefetch, read from disk using zHPF

The results show that with sequential prefetch, DS8800 channel throughput is 33% higher than the DS8300. Using dynamic prefetch, DS8800 channel throughput is 29% higher than the DS8300.

Prefetch reading larger pages from disk and cache

The fourth set of measurements was done with sequential and dynamic prefetch reading data from disk and cache, for page sizes of 4 and 16 KB. See Figure 3-14.

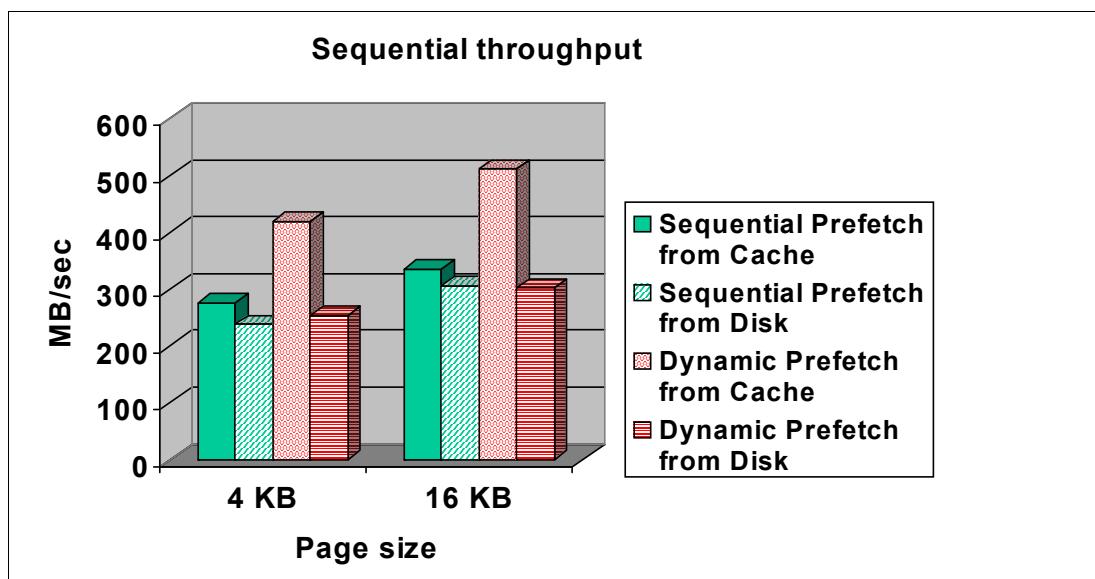


Figure 3-14 Sequential read from DS8800 disk by DB2

A larger page size of 16 KB increases the throughput by 19% to 28%.

When fetching from cache, dynamic is faster than sequential prefetch, but not so when fetching from disk (on the DS8800).

3.4.2 DB2 logging and insert with disk enhancements

These measurements were done on a z196, which enables the log I/Os exceeding 64 KB to be zHPF eligible, providing an additional boost in performance. Figure 3-15 shows that the DS8800 improves the maximum throughput by about 70% when the log buffer queues become large.

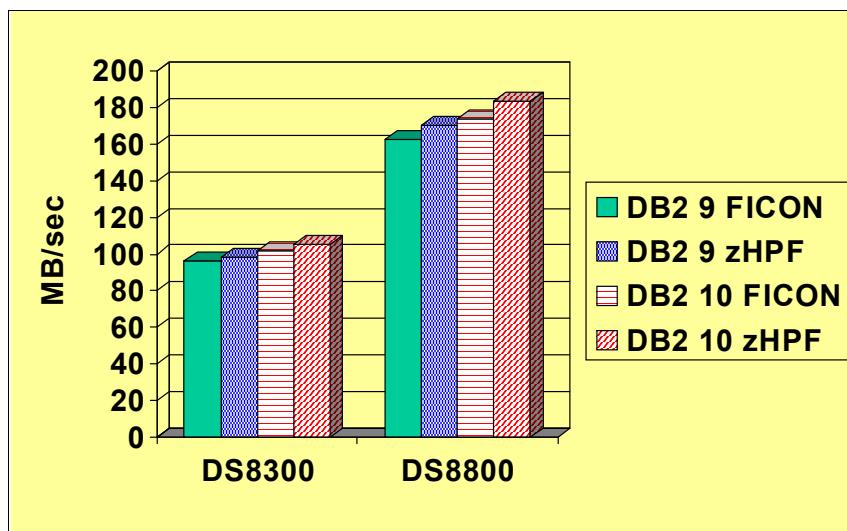


Figure 3-15 Maximum DB2 log throughput

If the number of 4 KB log buffers is less than or equal to 16, these writes are eligible for zHPF on a z10. If the number of buffers is greater than 16, the z10 cannot use zHPF, but the z196 can. Measurements show that DB2 10 can achieve a slightly higher log throughput with zHPF compared to FICON. z196 enables log I/Os greater than 64 KB to use zHPF and DS8800 requires FICON Express 8 to achieve 8 Gbps speed channels.

Load preformat is a convenient way to quantify preformat that results from insert processing. We measured write throughput both on the DS8300 and the DS8800. See Figure 3-16.

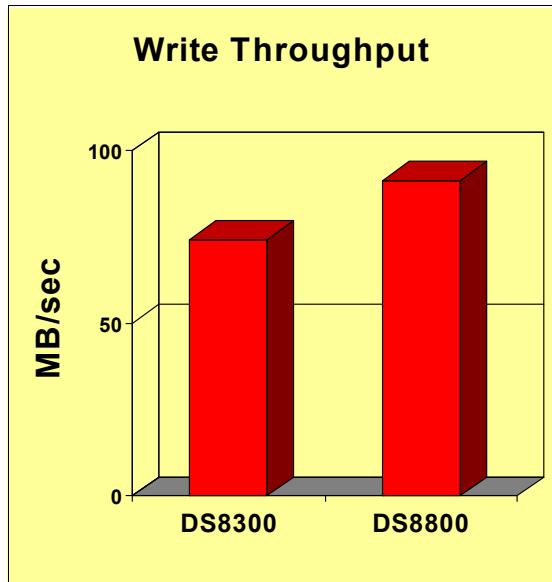


Figure 3-16 Load preformat

The results show that the DS8800 preformat throughput is 23% higher than the DS8300.

3.4.3 Utilities and storage enhancement

DB2 utilities use a variety of different types of I/Os. The z196 helps with sequential reads. The sequential reads, from a table space or index, are now zHPF-eligible. Format writes and list prefetch I/Os are not eligible. The sequential reads from DSORG=PS data sets are zHPF-eligible if the data set has extended format.

Performance measurements using DS8800 and zHPF were done for the following areas:

- ▶ Load utility
- ▶ Unload utility
- ▶ Utility BSAM enhancements

Load utility

The first set of measurements shows the Load utility performance using various page sizes on the DS8800 and the DS8300. See Figure 3-17.

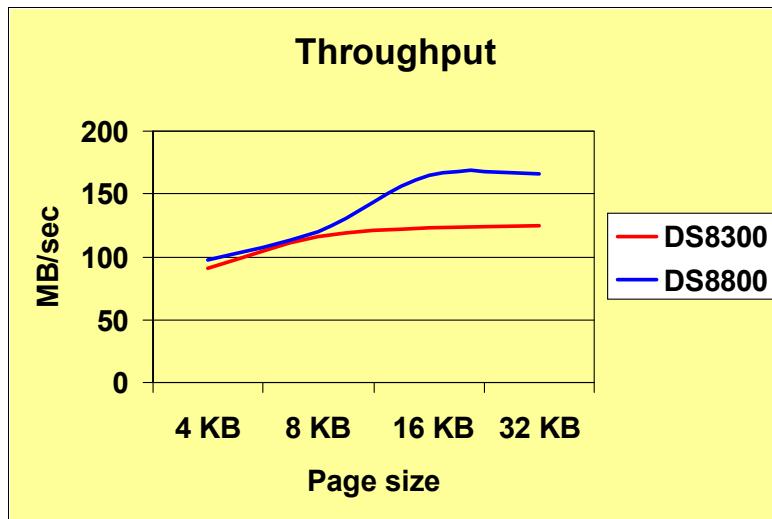


Figure 3-17 LOAD utility measurement results

The results show significant improvements with the 16 KB and 32 KB pages, but not with 4 KB and 8 KB pages because of the zHPF not supporting format writes.

The second set of measurements shows the LOAD utility performance when loading small 20 KB LOBs using 4 KB pages table. See Figure 3-18.

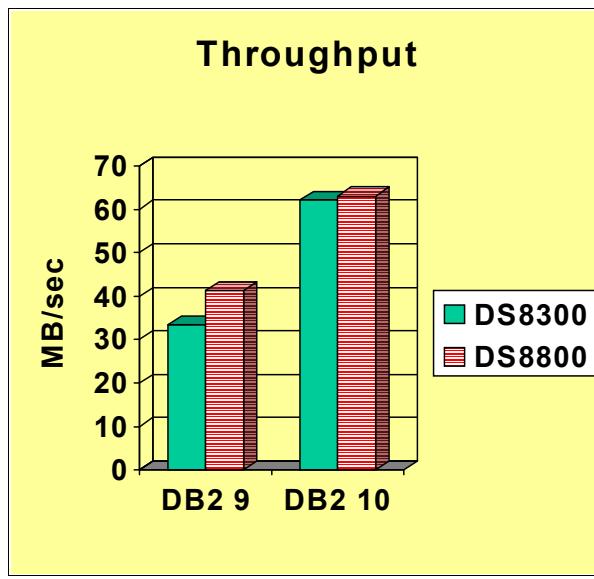


Figure 3-18 LOAD REPLACE of LOBs

The results show that with DB2 9, the DS8800 takes advantage of the format write. With DB2 10 both devices improve by about 50%, with the DS8800 not showing better performance than the DS8300 because DB2 is unable to drive the device utilization higher.

Unload utility

A set of Unload utility measurements were done with extended format data sets both on the DS8800 and the DS8800 in DB2 9 and DB2 10 as shown in Figure 3-19.

The DS8800 shows about 19% more throughput than the DS8300 in DB2 9 and 33% in DB2 10.

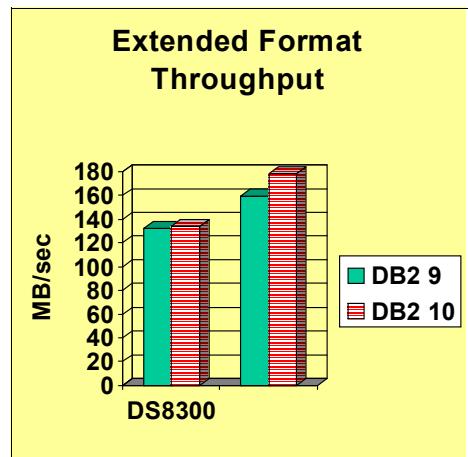


Figure 3-19 UNLOAD I/O throughput measurement

Utility BSAM enhancements

z/OS 1.9 introduced support for long-term page fixing of basic sequential access method (BSAM) buffers, and z/OS 1.10 introduced support for 64-bit BSAM buffers if the data set is in extended format. DB2 10 utilities exploit these recent z/OS enhancements, by offering the following enhancements:

- ▶ Allocating 64-bit buffers for BSAM data sets
- ▶ Allocating more BSAM buffers for faster I/O
- ▶ Long term page fixing BSAM buffers
- ▶ DB2 10 utilities, increasing MULTSDN² from 6 to 10 and MULTACC from 3 to 5

The measurements show how the value of DB2 10 and new storage enhancements benefit DB2 when making use of enhancements such as increasing MULTACC from 3 to 5, the faster HA and DA in the DS8800, zHPF, and multiple streams per RAID rank for EF and non-EF data sets.

² MULTSDN=n requests a system-defaulted NCP (number of channel programs).

Extended format BSAM data sets

Figure 3-20 shows the DS8800 performance using a block size of 27966 on extended format BSAM data sets. For the extended format, the zHPF can be enabled with a z196 processor.

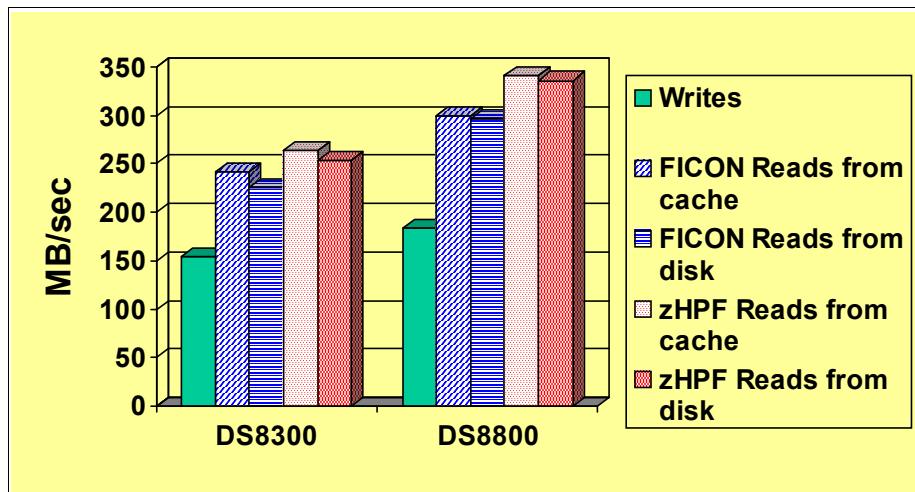


Figure 3-20 The EF BSAM measurements

The measurement results were more significant with the extended format, when comparing the DS8800 with the DS8300. The increase for write throughput was 20%. The FICON read channel throughput increased by 24%, and the disk throughput by 32% for the DS8800. In addition, the zHPF increased read channel throughput by 29% and disk throughput by 32% compared for the DS8800. On the DS8800, the increase for read throughput using the zHPF was 14%.

A measurement is done changing the number of the stream reading extended format BSAM data sets. Testing was done using a single rank, 5 tracks per I/O. See Figure 3-21.

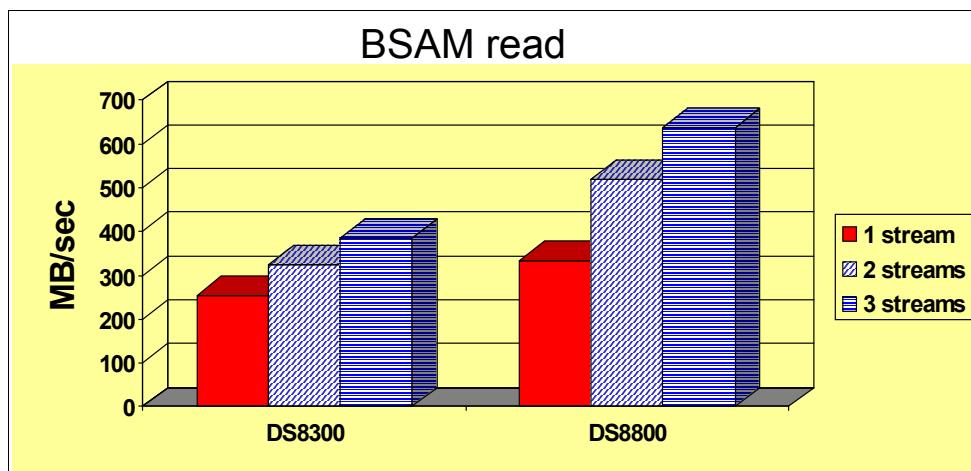


Figure 3-21 BSAM read changing the number of streams

The results show that a single rank DS8800 throughput with 1 stream was 32% higher than for DS8300. Using 2 streams, a single rank DS8800 throughput was 60% higher than for the DS8300. Likewise, using 3 streams, a single rank DS8800 throughput was 65% higher than for the DS8300.

The next case, shown in Figure 3-22, shows *write* measurements for an extended format BSAM data set, when changing number of streams.

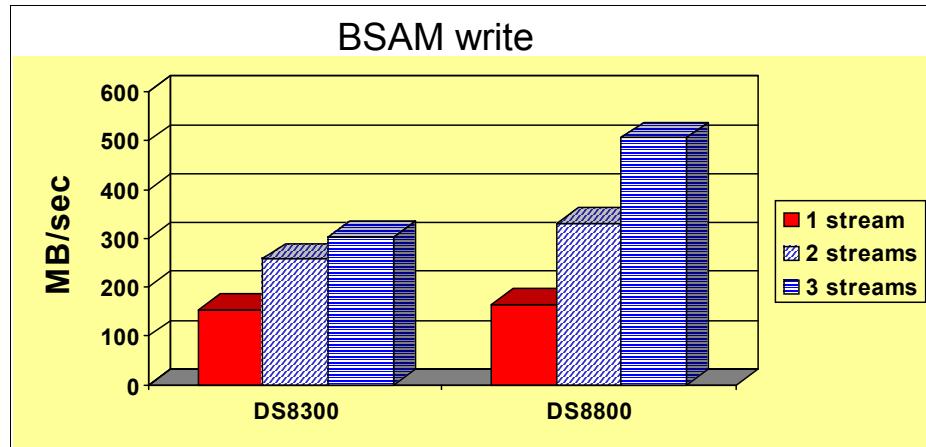


Figure 3-22 BSAM writes changing number of stream

The write case results are similar to the read case. One RAID rank scales better with 3 streams on a DS8800 than the DS8300, in comparison with the 1 or 2 streams results.

The DS8800 is the next chapter in the IBM flagship enterprise disk platform. Built on 50+ years of enterprise class innovation, the DS8800 enables much higher performance and scalability than its predecessor models, while preserving client investments in prior DS8000 models. The DS8800 offers faster processor speeds, faster adapters and buses, and all with a smaller footprint in terms of floor space and energy consumption. For DB2 this means that all workloads will run faster, and in this book we have illustrated faster prefetch I/O, faster log I/O and faster utilities. The DS8800 also provides for much better OLTP performance.

For more information, see IBM System Storage DS8800 Performance Whitepaper at this website:

<http://partners.boulder.ibm.com/src/atsmastr.nsf/WebIndex/WP101799>

3.4.4 DB2 support for solid state drives

A solid state drive (SSD) is a storage device that stores data on solid-state flash memory rather than traditional hard disks. SSDs contain electronics that enable them to emulate hard disk drives (HDDs). These drives have no moving parts and use high-speed memory, so they are fast and energy-efficient. They have been enhanced to be usable by enterprise-class disk arrays. These drives are treated like any HDD in the array.

SSDs can be used for allocating table space data sets or index data sets like HDD. The use of SSDs for enterprise storage is transparent to both DB2 and z/OS; however, because SSDs have no seek times, measurements have shown that SSDs can improve DB2 queries two to eight times as the result of a faster I/O rate over HDDs. There are other DB2 functions that are also affected by the drive type. The need to REORG to improve query performance is reduced when the table space is on SSD.

For more information about SSD, see *DB2 10 for z/OS: Technical Overview*, SG24-7892 and *Ready to Access DB2 for z/OS Data on Solid-State Drives*, REDP-4537.

Currently DB2 10 supports the following functions:

- ▶ DB2 10 tracks the device type on which the page sets reside. The column DRIVETYPE is added to the DB2 catalog tables SYSIBM.SYSTABLESPACESTATS and SYSIBM.SYSINDEXSPACESTATS.
- ▶ The stored procedure DSNACCOX is enhanced to consider the drive type that the data set is on when making REORG choices.

You can use the EasyTier feature (enabled with a microcode upgrade) to minimize the cost of managing the selection of “hot” data sets to store on the SSD.

With DS8800, which supports 300 GB SSD drives, the SSD maximum I/O rate for a fully configured CU is 156% higher than for the DS8300, and 49% higher than for the DS8700.

See the “IBM System Storage DS8800 Performance” white paper, available at:

- ▶ <http://partners.boulder.ibm.com/src/atスマstr.nsf/WebIndex/WP101799>
- ▶ http://www.ibm.com/systems/storage/disk/enterprise/ds_whitepapers.html

3.5 SMF compression

Today, the volume of data directed to SMF can be immense. DB2 10 introduces a new DSNZPARM parameter, SMFCOMP, which directs DB2 to request compression for trace records that are sent to SMF. Trace data to GTF and OPX is not compressed.

SMFCOMP is specified on the installation panel DSNTIPN in the field COMPRESS SMF RECS. The default value is OFF. See Figure 3-23.

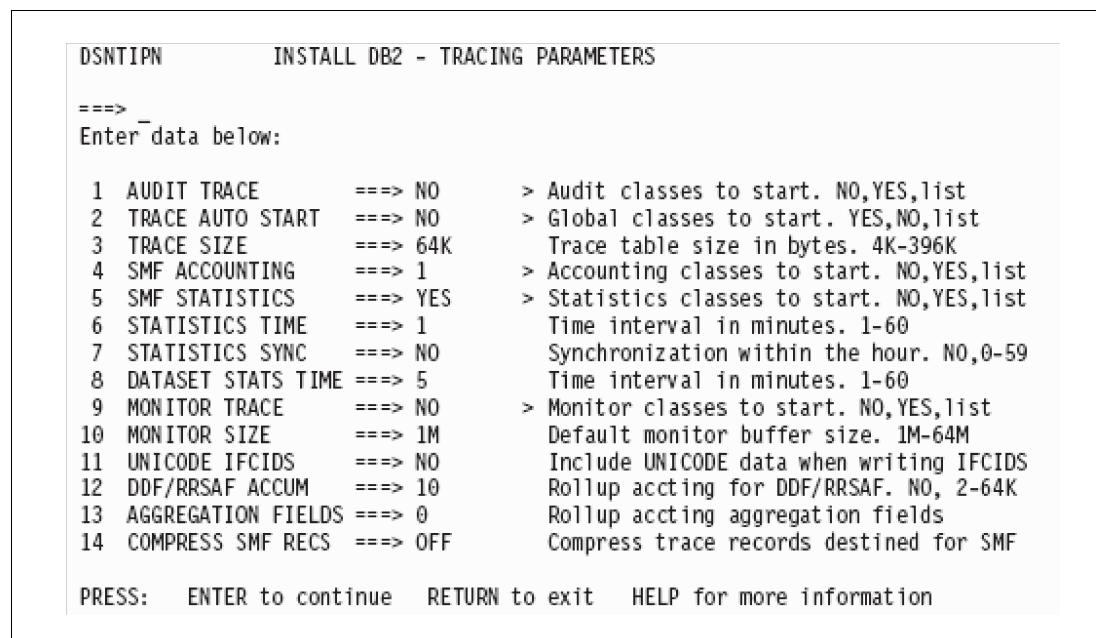


Figure 3-23 Tracing parameters panel DSNTIPN

If compression is enabled, SMF data is compressed such that everything after the SMF header (SM100END, SM101END, or SM102END) is compressed with z/OS compression service CSRICESRV. A compressed record is identified by a bit in the SMF100, 101, and 102 headers. The trade-off for this function is SMF volume versus an increase in CPU to compress and expand the records.

Performance measurements show a minimal overhead of up to 1% with accounting class 1, 2, 3, 7, 8, 10 active. The disk savings for DB2 SMF data set can be significant with compression rate of 60% to 80%. Our results showed statistics classes 1, 3, 4, 5, and 6 with compression rate around 60%.

If you use your own trace formatter, you need to call the z/OS compression service, which is turned off by default, to decompress the data. APAR PM27872 provides you with decompression routine DSNTSMFD and sample JCL, DSNTEJDS, to execute it. DSNTSMFD takes an SMF data as an input to decompress. DSNTSMFD gives you an output message of how much your data was compressed and the percentage saved by the compression. Figure 3-24 shows the output from DSNTSMFD with statistics traces records being compressed.

*** DSNTSMFD *** STARTING 2011/03/14 13:02:13		
<hr/>		
<hr/>		
Total records read:.....	2408	
Total DB2 records read:.....	1307	
Total DB2 compressed records read:.....	197	
Total DB2 compressed records decompressed:.....	197	
Total non-DB2 records read:.....	1101	
Aggregate size of all input records:.....	9237888	8M
Aggregate size of all input DB2 records:.....	2152260	2M
Aggregate size of all DB2 compressed records:....	170830	166K
Aggregate size of all output DB2 records:.....	2392486	2M
Aggregate size of all DB2 expanded records:....	411056	401K
Aggregate size of all non-DB2 input records:.....	7085628	6M
Percentage saved using compression.....	58%	
 Details by DB2 subsystem		
 ...		
Subsystem ID: DBOA		
Number of records:.....	259	
Number of compressed records:.....	197	
Aggregate size of DB2 records:.....	302890	295K
Aggregate size of DB2 compressed records:....	170830	166K
Aggregate size of DB2 expanded records:....	411056	401K
Percentage saved using compression.....	58%	
<hr/>		
*** DSNTSMFD *** ENDING 2011/03/14 13:02:13		

Figure 3-24 Sample DSNTSMPD output



Table space design options

With DB2 9 and DB2 10, several changes have impacted the way DBAs can deal with the physical design of data. DB2 is no longer just a relational database. We first had a wave of universal data base enhancements related to object orientation, and now we have support for new types of data, such as LOBs and XML.

The new table space organization, the universal table space, introduced with DB2 9, includes the best characteristics of partitioned and segmented. DB2 10 added the hash organization and inline LOBs for universal table spaces.

With the additional functions provided by DB2 10, the universal table space has moved in as the table space of choice.

In this chapter, we discuss the recent enhancements related to the following types of table spaces, concentrating on the their performance aspects:

- ▶ Universal table space
- ▶ XML
- ▶ Inline LOBs
- ▶ Hash access

4.1 Universal table space

Starting with DB2 9 for z/OS new-function mode (NFM), you can combine the benefits of segmented space management with partitioned table space organization using universal table spaces (UTS): either *partition-by-growth (PBG)* table spaces or *range-partitioned table spaces* (also known as *partition-by-range (PBR)* table spaces). The combined advantages are as follows:

- ▶ A segmented space-map page has more information about free space than a partitioned space-map page.
- ▶ Mass delete performance is improved because mass delete in a segmented table space organization tends to be faster than in other types of table space organizations.
- ▶ All or most of the segments of a table are ready for immediate reuse after the table is dropped or mass deleted.
- ▶ Partitioning allows for supporting of large table spaces and parallelism of accesses.

Also all table spaces, including UTS, are created in reordered row format by default, unless the DSNZPARM SPRMRRF is set to DISABLE.

The MEMBER CLUSTER option, introduced for partitioned table spaces in data sharing environments, is supported by UTS with DB2 10. For information about this function, see 4.1.5, “MEMBER CLUSTER option available for UTS” on page 80.

The DB2 catalog table SYSIBM.SYSTABLESPACE identifies table spaces by the following values in the TYPE column:

- | | |
|--------------|--|
| blank | The table space was created without the LOB or CLUSTER options |
| L | The table space can be greater than 64 gigabytes |
| R | UTS range-partitioned UTS |
| P | Implicit UTS table space created for XML columns |
| O | LOB table space |
| G | UTS partitioned-by-growth table space |

4.1.1 The use of UTS in DB2 9

In DB2 9, UTS was required for the following items:

- ▶ XML table spaces, but the base table can be segmented, partitioned, or UTS
- ▶ CLONE support

4.1.2 The use of UTS in DB2 10

In DB2 10, UTS is required if you want to use any of the following features:

- ▶ Hashing access. For information about this function, see 4.4, “Hash access”.
- ▶ XML versioning function. The XML *base table* need to be a UTS. For information about this function, see 4.2, “XML” on page 85.
- ▶ Use of ALTER online schema enhancements. Several schema changes are available for UTS with the new *pending changes* technique that are not available to the other structures of table spaces, as described in Chapter 4, “Availability” of the *DB2 10 for z/OS Technical Overview*, SG24-7892.

- ▶ Inline LOBs. For information about this function, see 4.3, “Inline LOBs” on page 95.
- ▶ Access currently committed data. For information about this function, see 7.8, “Access currently committed data” on page 230.
- ▶ Insert index I/O parallelism. This requires either UTS or a classic partitioned table space but does not work with segmented table spaces. See 2.7, “I/O parallelism for index updates” on page 43.

4.1.3 How to convert to UTS

With DB2 9, you cannot convert a table space to UTS without a drop and re-create. DB2 10 largely simplifies changes to table space structure and attributes. Figure 4-1 shows the ALTER supported table space type conversions, which are as follows:

- ▶ Convert index partitioned table space to table partitioned.
- ▶ Convert classic table partitioned table space to a range-partitioned table space adding SEGSIZE.
- ▶ Convert simple table space with one table to a partition-by-growth table space.
- ▶ Convert segmented table space with one table to a partition-by-growth table space.
- ▶ Convert a partition-by-growth table space to hash table space.
- ▶ Convert a range-partitioned table space to hash table space.

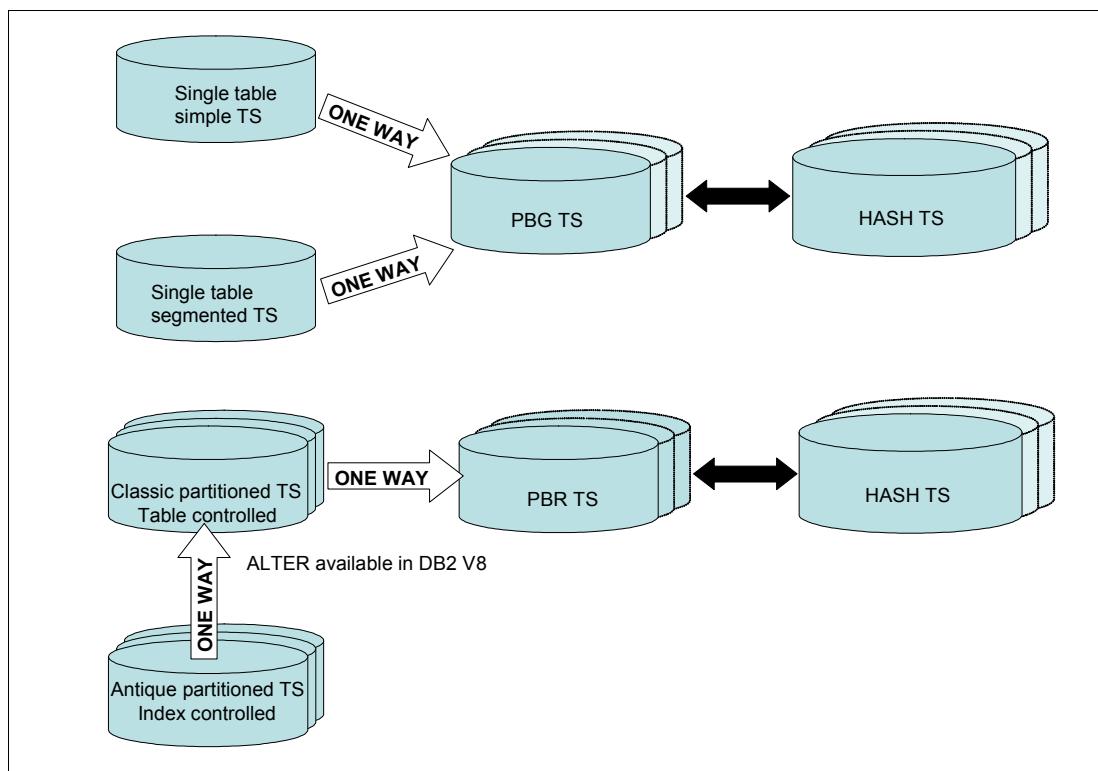


Figure 4-1 Possible table space type conversions

Note that these conversions are only allowed for single-table table spaces, and a conversion back from hash table spaces does not use the pending changes technique. See *DB2 10 for z/OS Technical Overview*, SG24-7892.

4.1.4 New default table space at CREATE time

In DB2 10, when creating a new table space, segmented table spaces are no longer the default. Partition-by-growth universal table spaces are the new default.

If you want to create a range partitioned UTS, set values for NUMPARTS and SEGSIZE.

If you want to create a classic partitioned table space, you need to specify SEGSIZE 0 on CREATE TABLESPACE. You can also set the new default partition SEGSIZE DSNZPARM DPSEGSZ=0 in the DSN6SYSP macro (the default is 32).

If you want to create a segmented table space, do not specify MAXPARTITIONS and NUMPARTS, and whatever the DPSEGSZ value, you get a segmented table space with 4 KB SEGSIZE.

4.1.5 MEMBER CLUSTER option available for UTS

When you INSERT a row, DB2 typically tries to place data in the clustering sequence as defined by the implicit clustering index (the first index created) or the explicit clustering index. This can cause “hot spots” in data page ranges and high update activity in the corresponding space map page or pages. These updates must be serialized among all members in a data sharing environment, which can adversely affect INSERT/UPDATE performance in data sharing.

DB2 Version 5 introduced the MEMBER CLUSTER option on CREATE TABLESPACE statement of a partitioned table space to address this performance issue. The MEMBER CLUSTER option causes DB2 to manage space for inserts on a member-by-member basis instead of by using one centralized space map. The main idea of a MEMBER CLUSTER page set is that each member has exclusive use of a set of data pages and their associated space map page. (Each space map covers 199 data pages.) Each member will INSERT into a different set of data pages and not share them, thereby eliminating contention on pages and reducing time spent searching for pages. The MEMBER CLUSTER option has been used successfully to reduce contention in heavy INSERT applications.

DB2 9 introduced universal table spaces (UTS) and several functions are supported only through UTS. However, DB2 9 does not support MEMBER CLUSTER for UTS. So, you have to choose between the benefits of MEMBER CLUSTER (reduced contention in a heavy INSERT environment) with the flexibility of UTS (better space management and new functions).

DB2 10 in new-function mode removes this restriction. MEMBER CLUSTER is supported by both partition-by-growth and range-partitioned UTS and it can be created as shown:

- ▶ To create a MEMBER CLUSTER partition-by-range UTS, use these statements:

```
CREATE TABLESPACE MySpace IN MyDB  
MEMBER CLUSTER  
NUMPARTS 3;
```

- ▶ To create a MEMBER CLUSTER partition-by-growth UTS, use these statements:

```
CREATE TABLESPACE MySpace in MyDB  
MEMBER CLUSTER  
MAXPARTITIONS 10;
```

- You can also implement MEMBER CLUSTER using an ALTER, without having to drop and recreate the table space. Because the need to use MEMBER CLUSTER can change over time, DB2 10 also provides the capability to both enable and disable MEMBER CLUSTER for universal table spaces:

```
ALTER TABLESPACE MyDB.MySpace ..... MEMBER CLUSTER YES/NO
```

There is an optional algorithm in DB2 9 with MEMBER CLUSTER that optimizes INSERT performance (reduced space search) while at the same time it tries to reuse disk space at a moderate level. To enable this algorithm, a partitioned table space needs to be created with MEMBER CLUSTER and FREEPAGE = 0 and PCTFREE = 0. It is often used in SAP for crucial tables. With DB2 10, this option for INSERT algorithm that relies on MEMBER CLUSTER and FREEPAGE = 0 and PCTFREE = 0 is also available for UTS table spaces.

As with prior versions of DB2, a MEMBER CLUSTER table space becomes unclustered quickly. You need to REORG the table space to bring the data back into clustering sequence if needed.

See also 5.3.2, “Insert performance measurements” on page 150 for measurements with MEMBER CLUSTER.

4.1.6 UTS workload performance

In this section we report some performance observations useful when choosing whether or not to use a universal table space (UTS). The focus here is on the performance of highly concurrent insert with data sharing, which is usually the most critical workload, where we have several performance measurements to use as a guide. From the functionality point of view, you can see what UTS provides in 4.1.2, “The use of UTS in DB2 10” on page 78. For impact on a simple OLTP environment, such as the IRWW workload, see “IRWW OLTP workload with UTS” on page 130.

Currently segmented is the only table space type that supports multiple tables, but it is likely that highly concurrent insert will not occur when lots of small tables are sharing a segmented table space. Segmented is also the only table space type in DB2 10 that is limited to 64 GB (besides simple table spaces, which are still supported but can no longer be created). Furthermore, segmented is the only table space type in DB2 10 that does not support member cluster (MC), which is an important feature for optimizing high concurrent insert performance with data sharing.

In our discussion we divide the table space types into two *classes*:

- The first class, for range defined table spaces, includes classic partitioned table spaces (PTS) and UTS partition by range (PBR).
- The other class, for non-range-defined table spaces, includes segmented (SEG) and UTS partition-by-growth (PBG).

We use these abbreviations in the measurements’ charts.

We do not directly compare the performance of the two classes to each other. They fulfill different application requirements.

For each class, we consider both *random* inserts and *sequential* inserts. Sequential meaning that the rows are appended at the end of the table because no free space exists.

We consider both *page level locking* (PLL) and *row level locking* (RLL). We expect RLL to perform worse than PLL when doing sequential inserts. Likewise, we expect RLL to perform worse than PLL when a query does sequential reads.

Let us consider why you might or might not want to use *member cluster*. See also 4.1.5, “**MEMBER CLUSTER** option available for UTS” on page 80. The purpose of member clustering is to improve concurrent insert performance in a data sharing environment.

However, member clustering reduces the amount of clustering, which in turn can hurt the performance of queries, especially when using the cluster index to locate rows. If clustering matters, then you might want to avoid member cluster.

But, if clustering matters, then your queries are tending to access the rows sequentially, and in that case, the RLL performance penalty is significant. So, if clustering matters, to optimize performance, you need to use PLL.

On the other hand, PLL is more likely than RLL to cause lock contentions, timeouts, and deadlocks. Hence, if clustering matters, you might need to make a tradeoff between optimal performance and minimizing deadlocks.

Figure 4-2 shows the environment setup.

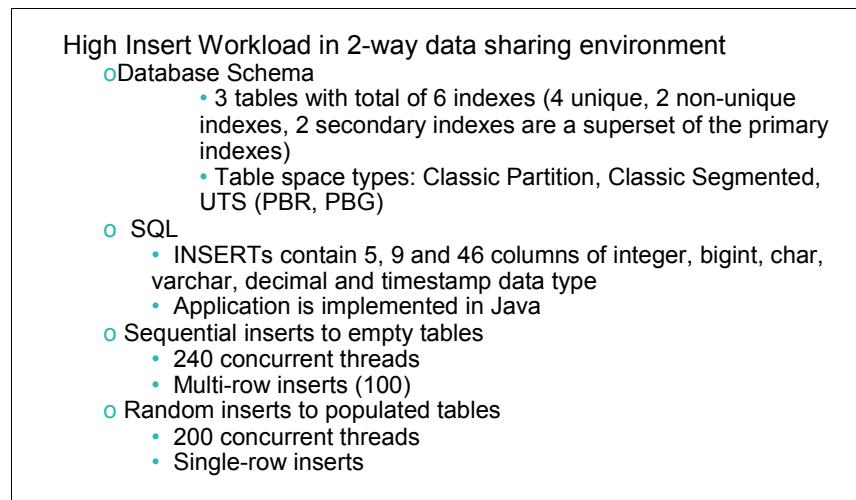


Figure 4-2 Workload environment definition

Now we are ready to examine the high performance insert measurements. As mentioned, we first look at table spaces:

- ▶ Non-range-defined table spaces
- ▶ Range defined table spaces

Non-range-defined table spaces

Figure 4-3 shows the performance results for the segmented and UTS partitioned by growth, with and without member cluster, for random and sequential inserts and both page level locking and row level locking.

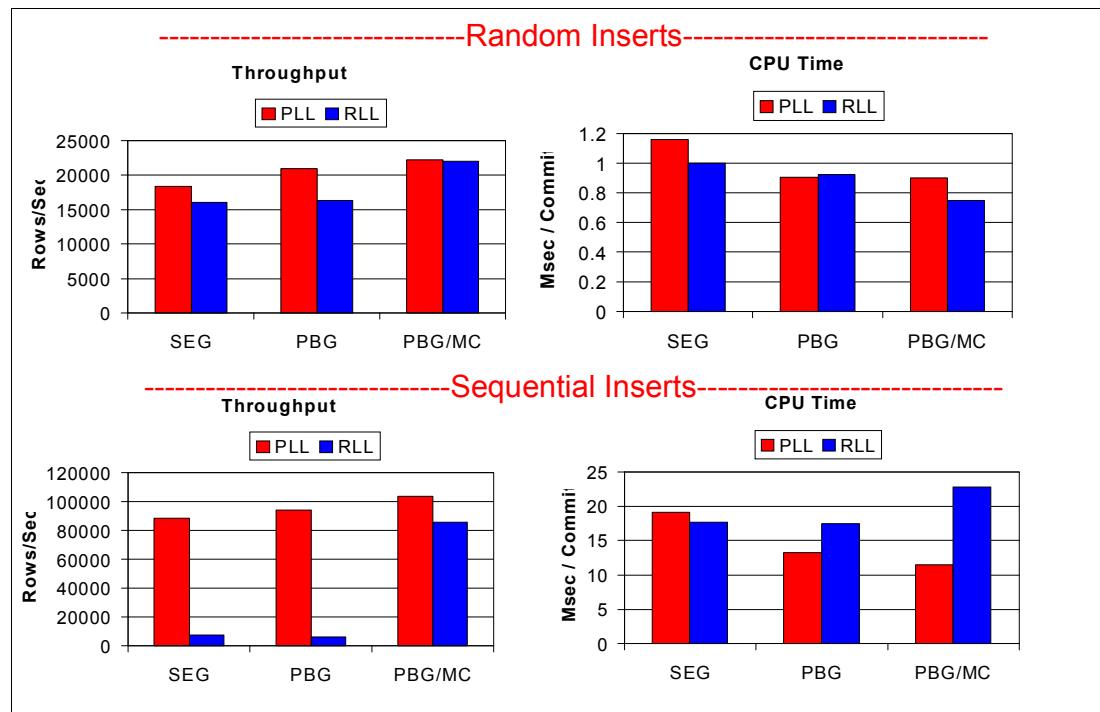


Figure 4-3 Non-range defined table spaces

If clustering does not matter, then PBG/MC offers the best performance both in terms of throughput and CPU time. For example, for random inserts with RLL, PBG/MC provides 37% more throughput than SEG with 25% less CPU time.

PBG and PBG/MC also perform better than SEG for sequential inserts provided that PLL is used. The questionable case is RLL with sequential access, because it requires a tradeoff. PBG/MC with RLL uses 29% more CPU time than SEG, but it has 12 times more throughput than SEG. PBG/MC uses 30% more CPU than PBG without MC, but again it has 12 times more throughput. Thus, for high insert throughput PBG/MC is the best choice, but if you do not want to sacrifice CPU performance for high throughput, and if you cannot use PLL, then you need to consider using PBG or SEG without MC.

Now let us ignore MC. If you want to use RLL, and if the inserts are sequential, then the CPU time of PBG and SEG are roughly equivalent, but SEG offers 12% more throughput than PBG. On the other hand, with random inserts and RLL, PBG and SEG have roughly the same throughput, while PBG uses 8% less CPU than SEG.

Range defined table spaces

The performance results for the partitioned and UTS partitioned by range, with and without member cluster, for random and sequential inserts and both page level locking and row level locking, are shown in Figure 4-4.

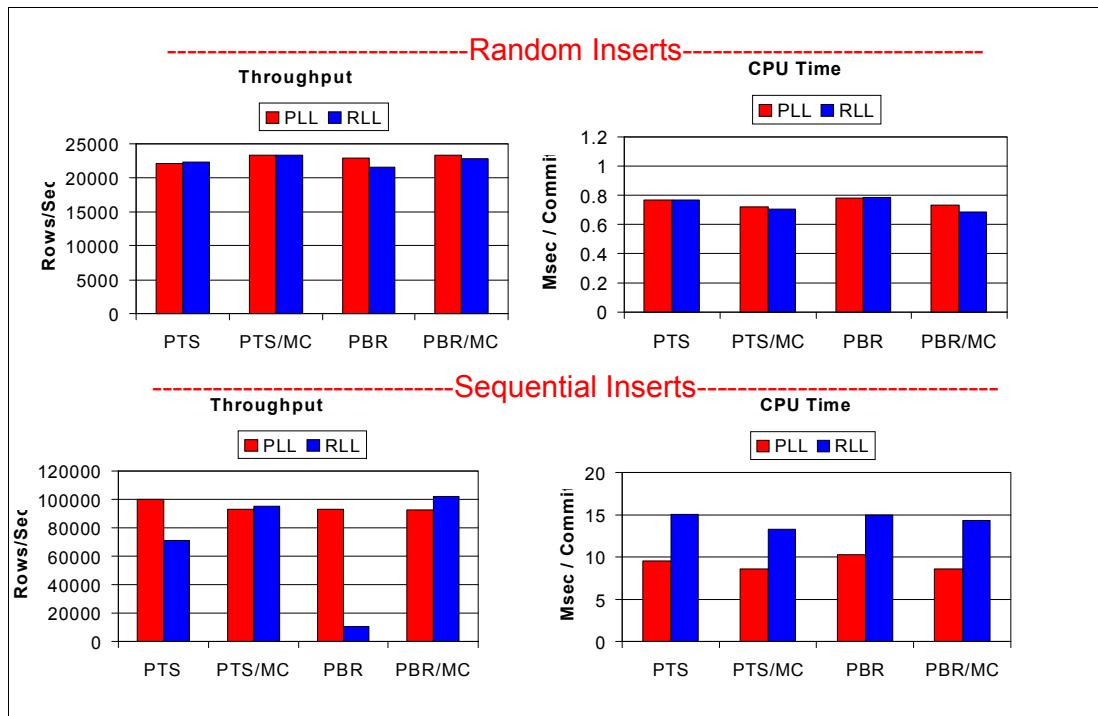


Figure 4-4 Range defined table spaces

With random inserts, there is not a lot of difference between PBR and PTS. Whether we use PBR or PTS, we see that MC shows a small improvement over non-MC. With PLL without MC, PTS is 7% to 8% better than PBR in both throughput and CPU time. With RLL, that difference becomes 3% to 4%.

Now let us look at sequential inserts. Using member cluster and PLL, PBR and PTS perform the same way. With PLL without member cluster, PTS is 7% to 8% better than PBR in terms of throughput and CPU time. Now let us turn to RLL. With member cluster, PBR attains 8% more throughput than PTS, but it uses 8% more CPU. So the choice is a tradeoff between throughput and CPU time. There is a more obvious problem with PBR if MC cannot be used with RLL; although the CPU time of PBR is equivalent to PTS, the throughput is 7 times less.

The PBR throughput problems with RLL are unique to a data sharing environment. When running the same high concurrent insert workload for sequential inserts on a single DB2 member, PTS and PBR performed the same.

Considerations on UTS performance

In conclusion, in many cases, universal table spaces provide you with optimal high concurrent insert performance in a data sharing environment, especially if you can take advantage of MC with PBG.

Even if MC is undesirable, PBG gives you good performance. PBR also gives you very good performance if PLL locking is used, or if your inserts are random.

The one case where universal table space performance might suffer significantly is PBR sequential inserts with row level locking in a data sharing environment, where MC is undesirable. If high concurrent insert throughput is important to you, and if you do not want to use MC, and if you want to use the new functions that universal table spaces provide, consider to avoid row level locking.

Inferences to hash tables

Suppose that you are considering to use the hash access organization (see 4.4, “Hash access” on page 112). In order to consider a hash organization, you must forego a cluster index. If you start with an empty table and then populate the table using inserts, there is plenty of free space in the hash table and there is no need to search for space. Those inserts are *random* because the hash values are pseudo random. After the hash table starts to become say 80% to 90% full, the inserts begin to use the overflow area more frequently, at which point those overflow inserts might start to behave in a sequential manner in the same manner as ordinary tables.

Inferences to inline LOBs

Suppose you are considering to use inline LOBs (see 4.3, “Inline LOBs” on page 95). Inline LOBs cause your row size to grow, meaning that there are fewer rows per page. When there is only one row per page, RLL and PLL are identical except in data sharing where RLL requires a data page P-lock. In the following measurements you can observe a large RLL performance penalty, but that is not the case when the rows are large. So, if you want to understand the performance of large rows, consider the PLL measurements even if you plan to use RLL.

4.1.7 Summary for universal table spaces

In DB2 10, you can convert to universal table spaces (UTS) using ALTER and REORG. After the table space is converted to UTS, you can also change many other attributes, such as page size, or take advantage of new functions that require UTS. Thus, to be able to alter these other attributes, create all *new* table spaces as UTS, and as time and resources permit, convert the existing table spaces to UTS.

The INSERT performance of the UTS is close to the classic partition table space and better than the classic segmented. MEMBER CLUSTER shows some benefits in data sharing.

In the case of UTS sequential insert with row level locking in data sharing, the use of the MEMBER CLUSTER option can be more beneficial.

In summary, in DB2 10 with UTS, and MEMBER CLUSTER support, you can get performance *generally* equivalent to classic partitioned table spaces and better than segmented.

4.2 XML

Prior to DB2 9 for z/OS, the only way for you to store XML data in DB2 was as string data and you had to use the XML Extender capabilities to parse an XML document stored in DB2. Starting with DB2 9, you can store XML data using the XML data type and use the DB2 pureXML function to store, process, and manage XML data in its native hierarchical format.

A component of DB2 pureXML is SQL/XML, which is an extension to the SQL standard as defined by ISO/IEC 9075-14:2003. Because SQL/XML is standards-based, it is supported by many other databases. It provides a range of XML related extensions to the SQL language, to allow XML data to be accessed.

The functions provided by SQL/XML can be categorized into the following main groups:

- ▶ XML publishing functions, which allow XML documents to be created from the contents of relational data

- ▶ XML handling functions, which allow the user to embed XPath expressions in SQL statements. XPath expressions are a subset of the XQuery standard.
- ▶ XML conversion functions, which support the interchange of data between relational and XML models of data

DB2 10 provides many enhancements to the performance of XML processing, with several of these performance enhancements also retrofitted to DB2 9 by DB2 maintenance. Informational APAR II14426 describes all XML APARs, including the performance enhancements. In the following sections we describe the performance enhancements to XML in DB2 10, along with some measurements.

For more information about XML and DB2 for z/OS, see *Extremely pureXML in DB2 10 for z/OS*, SG24-7915.

These are the main performance enhancements that are available in DB2 10:

- ▶ XMLTABLE shredding enhancements: Benefits XMLTABLE queries with many columns (up to 47% improvement on some XMLTABLE queries with many columns)
- ▶ Complex XPath predicate enhancements: Benefits XPath with “and” and “or” predicate conditions (up to 40% improvement on XPath with complex predicates)
- ▶ XML new function mode enhancements: Binary XML, XML date/time indexability, XML Schema validation “inside the engine”

The following XML topics are covered in this section:

- ▶ XML transaction processing performance
- ▶ Modifying part of an XML document
- ▶ Indexes on XML DATE and TIMESTAMP data
- ▶ XML schema validation
- ▶ XML type modifier
- ▶ Support for binary XML
- ▶ Support for multiple versions of XML documents

4.2.1 XML transaction processing performance

In order to test the performance enhancements in DB2 10, we ran a securities trading benchmark using an XML-only database. The test was based on the FIXML protocol, which is the standard XML schema for the finance industry. FIXML is an abbreviation for Financial Information Exchange Markup Language.

There were three components to this XML transaction processing workload:

- ▶ Massive insert
- ▶ Query
- ▶ Mixed transaction (select, insert, update, delete)

Our performance measurements for this workload are as follows:

- ▶ Massive insert: DB2 10 class 2 CPU time is 2.25% better than DB2 9.
- ▶ Query: DB2 10 class 2 CPU time is 2.78% better than DB2 9.
- ▶ Mixed transaction: DB2 10 class 2 CPU time is 5.6% better than DB2 9.

These CPU improvements are on top of the performance enhancements that were retrofitted to DB2 9 and already produced a 2 times general performance improvement for each of the workloads.

4.2.2 Modifying part of an XML document

In DB2 9, if you make modifications to part of an XML document stored in a DB2 table, the application cannot specify the required modification to the XML document. Applications that require parts of XML documents to be modified need to break apart the XML document into modifiable pieces, make the modification to a piece of the XML document, and then construct the pieces back into a single XML document.

XMLMODIFY

DB2 10 includes support for updating part of an XML document by introducing the scalar function XMLMODIFY. This new function, sometimes called a sub-document update, allows you to modify portions of documents; only the affected records are updated. The goal of this new function is to perform the same or better than doing a full-document update. A sample UPDATE statement using the XMLMODIFY function is shown in Example 4-1.

Example 4-1 Sample use of XMLMODIFY function for sub-document UPDATE

```
UPDATE USRT014.CUSTACC SET CADOC=
    XMLMODIFY('declare default element namespace
        "http://tpox-benchmark.com/custacc";
        replace value of node
        /Customer/CountryOfResidence
        with $value',
        'Iraq' AS "value")
    WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)
```

In this sample UPDATE statement, DB2 updates only the /Customer/CountryOfResidence node with the value of 'Iraq'. There is no need to replace the entire document.

XMLMODIFY performance

We ran a number of UPDATE statements using XMLMODIFY to compare the performance of a partial update to an equivalent UPDATE statement that will do a full update of the XML document. We tested the following variations of partial update:

- ▶ Insert after <node>
- ▶ Insert before <node>
- ▶ Insert as first <into node>
- ▶ Insert as last <into node>
- ▶ Delete node (s) - Delete one
- ▶ Delete node (s) - Delete many
- ▶ Replace <node>
- ▶ Replace value of <node>

The nine SQL statements (eight for partial updates as just listed and one for full update) that we used for our test are shown in Example 4-2 and Example 4-3. The SQL statements to perform the full update and to perform the INSERT NODE variations are shown in Example 4-2.

Example 4-2 UPDATE statements using XMLMODIFY to test partial update - part 1

Full doc update:

```
UPDATE USRT014.CUSTACC SET CADOC = CAST (?) AS XML
    WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)
```

Sub-doc updates:

```
UPDATE USRT014.CUSTACC SET CADOC=
```

```

XMLMODIFY('declare default element namespace
           "http://tpox-benchmark.com/custacc";
           insert node $node as last into
           /Customer/Addresses',
           XMLELEMENT(NAME "NewAddr", 'inserted as last') AS "node")
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)

UPDATE USRT014.CUSTACC SET CADOC=
  XMLMODIFY('declare default element namespace
             "http://tpox-benchmark.com/custacc";
             insert node $node as first into
             /Customer/Addresses',
             XMLELEMENT(NAME "NewAddr", 'inserted as first') AS "node")
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)

UPDATE USRT014.CUSTACC SET CADOC=
  XMLMODIFY('declare default element namespace
             "http://tpox-benchmark.com/custacc";
             insert node $node before
             /Customer/Addresses',
             XMLELEMENT(NAME "NewAddr", 'inserted before') AS "node")
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)

UPDATE USRT014.CUSTACC SET CADOC=
  XMLMODIFY('declare default element namespace
             "http://tpox-benchmark.com/custacc";
             insert node $node after
             /Customer/Addresses',
             XMLELEMENT(NAME "NewAddr", 'inserted after') AS "node")
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)

```

The SQL statements to perform the DELETE NODE and REPLACE NODE variations are shown in Example 4-3.

Example 4-3 UPDATE statements using XMLMODIFY to test partial update - part 2

More sub-doc updates:

```

UPDATE USRT014.CUSTACC SET CADOC=
  XMLMODIFY('declare default element namespace
             "http://tpox-benchmark.com/custacc";
             delete node
             /Customer/Gender')
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)

UPDATE USRT014.CUSTACC SET CADOC=
  XMLMODIFY('declare default element namespace
             "http://tpox-benchmark.com/custacc";
             delete node
             /Customer/Addresses/Address')
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)

UPDATE USRT014.CUSTACC SET CADOC=
  XMLMODIFY('declare default element namespace
             "http://tpox-benchmark.com/custacc";
             replace node
             /Customer/CountryOfResidence')

```

```

        with $node',
        XMLELEMENT(NAME "CountryOfResidence",
        XMLNAMESPACES(DEFAULT
'http://tpox-benchmark.com/custacc'),
        'USA') AS "node")
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (? AS INTEGER)
UPDATE USRT014.CUSTACC SET CADOC=
XMLMODIFY('declare default element namespace
"http://tpox-benchmark.com/custacc";
replace value of node
/Customer/CountryOfResidence
with $value',
'Iraq' AS "value")
WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (? AS INTEGER)

```

Small size documents

The performance measurements comparing a full document update to various sub-document updates for small documents are shown in Figure 4-5.

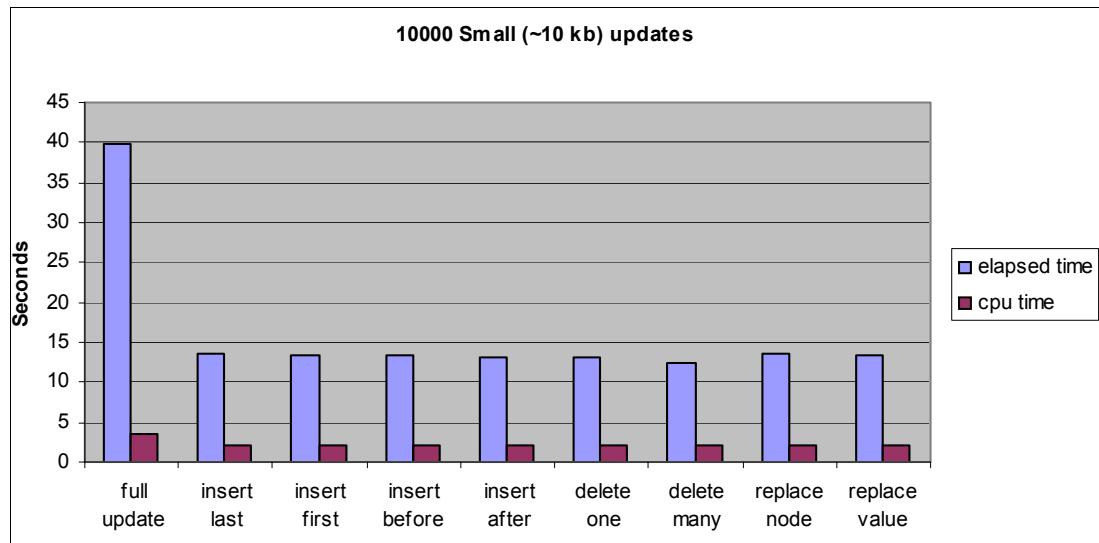


Figure 4-5 XMLMODIFY performance measurements for small documents

The first pair of bars represent the cost of doing a full update, which was the only option in DB2 9. The subsequent pairs of bars represent the cost of doing various insert, update or delete operations on a portion of the document using XMLMODIFY, which became available in DB2 10. The savings are between 66% and 69% for elapsed time and between 39% and 44% for CPU time.

The savings depend on the size of the documents and the complexity of the SQL statement. The measurements in Figure 4-5 were for simple XML updates of small documents. The SQL statement in Example 4-4 uses XMLMODIFY to do a sub-document update, but it inserts a whole document into the existing document. The statement performs host variable materialization and XML parsing. This SQL statement performs worse than a full update of the existing document. The elapsed time is 7% higher and the CPU time is 23% higher.

Example 4-4 Sample use of XMLMODIFY that is poorly performing

```
UPDATE USRT014.CUSTACC SET CADOC=
    XMLMODIFY('declare default element namespace
        "http://tpox-benchmark.com/custacc";
        insert node $node as last into
        /Customer/Addresses',
        CAST (? AS XML) AS "node")
    WHERE DB2_GENERATED_DOCID_FOR_XML = CAST (?) AS INTEGER)
```

XMLMODIFY is beneficial for simple updates, because the less data that gets updated the greater the savings of updating just that small amount of data rather than the whole document.

Medium size documents

The savings are even greater when we tested simple updates against medium sized documents. Figure 4-6 shows the CPU and elapsed time comparisons for a full document with the same variations of XMLMODIFY for a medium sized document.

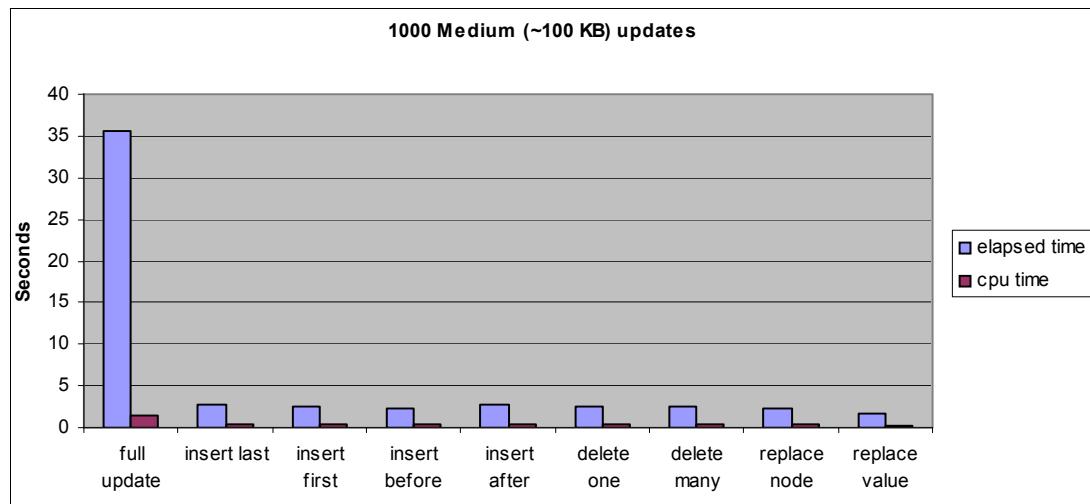


Figure 4-6 XMLMODIFY performance measurements for medium documents

The elapsed time savings are between 92% and 95%, while the CPU savings are between 72% and 84%.

Large size documents

The performance measurements for using XMLMODIFY to update large documents are shown in Figure 4-7.

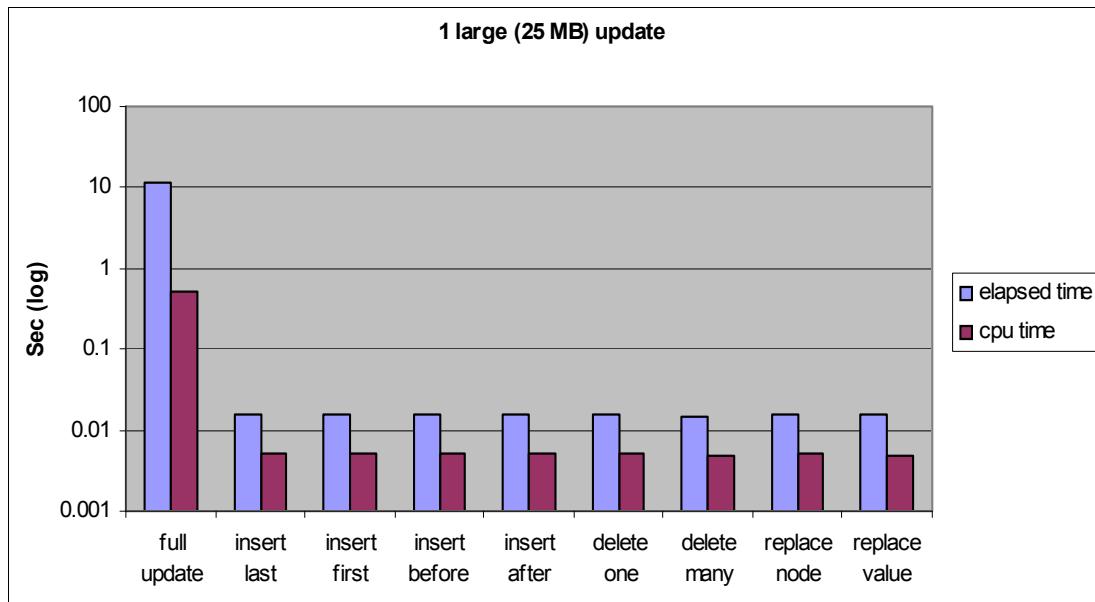


Figure 4-7 XMLMODIFY performance measurements for large documents

Note that the savings for using XMLMODIFY to update part of a document is much greater than the savings for small or medium sized documents. The scale for the seconds (y-axis) is logarithmic because of the difference in times for the full update versus the sub-document updates. The elapsed time and CPU time savings are over 99% for all the sub-document update tests compared to the full document update test.

The use of the XMLMODIFY function to update a portion of an XML document can provide significant savings over doing a full update of the document. XMLMODIFY provides the biggest savings for simple updates of large documents, because those types of updates provide the greatest difference between the amount of data actually being updated and the size of the document. Do not use XMLMODIFY to update an entire document, because the cost will be greater than if you did a full document update.

The XMLMODIFY function is available in new function mode and is only available for XML columns in which the base table has been created in a universal table space (UTS) after you migrate to DB2 10 new function mode (NFM). The reason for this requirement is that sub-document updates need to use multiple versions of XML data and the multi-versioning support is only available for tables that have been created in a UTS after migrating to DB2 10 NFM. If the XML column existed in a UTS prior to migrating to DB2 10 new function mode and you want to take advantage of this feature, then you need to drop and recreate the table to allow it to use the multi-versioning support. See 4.2.7, “Support for multiple versions of XML documents” on page 95 for details.

XML fixes: APAR PM28385 (PTF UK66136) provides several XML fixes including improvement to XMLMODIFY performance.

4.2.3 Indexes on XML DATE and TIMESTAMP data

Prior to DB2 10, you can index portions of an XML document, but the index can only be defined with a data type of VARCHAR or DECFLOAT. Starting with DB2 10, you can now create an index on XML data using a data type of DATE or TIMESTAMP. A sample CREATE INDEX statement to index DATE data is shown in Example 4-5.

Example 4-5 Sample DDL to create index on XML data using DATE data type

```
CREATE INDEX ORDERDAT ON ORDER(ODOC)
  GENERATE KEY USING XMLPATTERN
    'declare default element namespace
     "http://www.fixprotocol.org/FIXML-4-4";
    /FIXML/Order/orderdate'
  AS SQL DATE
```

The sample DDL creates an index named ORDERDAT on the element *orderdate* and specifies that the index will be stored as a DATE. Queries that search for rows with a specific order date in the XML document will now be able to use an index and perform DB2 DATE logic based on the value of the order date.

The performance of queries that use an index on a DATE data type is similar to the other supported index data types. In addition you now have the flexibility to perform date operations on elements that have a DATE or TIMESTAMP index defined.

Additional indexes cause overhead for INSERT operations. The overhead for a DATE or TIMESTAMP index on an XML document is similar to the overhead for VARCHAR indexes.

Indexes on XML DATE and TIMESTAMP data are available in new-function mode. Multi-versioning support is not required.

4.2.4 XML schema validation

XML schema validation is the process of determining whether the structure, content, and data types of an XML document are valid according to an XML schema. In addition, XML schema validation strips ignore white space from the input document.

Validation with DB2 9

In the base code of DB2 9, XML schema validation is done by invoking the user-defined function SYSFUN.DSN_XMLVALIDATE, which must be invoked within the XMLPARSE function. This validation requires setting up and administering the WLM application environment. You cannot take advantage of DB2 DRDA zIIP or zAAP redirect due to the user-defined function.

Validation with DB2 10

In DB2 10, and in DB2 9 after APARs PK90032 and PK90040 have been applied, XML schema validation is done inside the engine using the z/OS XML system service. You can specifically request schema validation by calling the new built-in function SYSIBM.DSN_XMLVALIDATE. In addition, if you have an XML type modifier defined on the XML column, you can expect the same performance for the validation process as if you called the SYSIBM.DSN_XMLVALIDATE function.

This enhancement allows the validation of XML documents over 50 MB and also allows the XML parser process invoked through XML validation to take advantage of IBM Specialty Engines (zIIP or zAAP). Each of these enhancements can result in a reduction in class 1 CPU time.

XML schema validation performance

To test the performance of the XML schema validation enhancements in DB2 10, we ran tests to measure the XML schema validation cost for various sized documents in DB2 9 (without the XML schema APARs applied) and in DB2 10.

The measurements that we performed were for the following types of workloads and document sizes, based on industry standard workloads:

- ▶ Custacc (4-18 KB document size)
- ▶ Medical (10 MB document size)
- ▶ SEPA (1 MB, 10 MB and 25 MB document sizes)
- ▶ Govt (25 MB document size)

The results are shown in Figure 4-8.

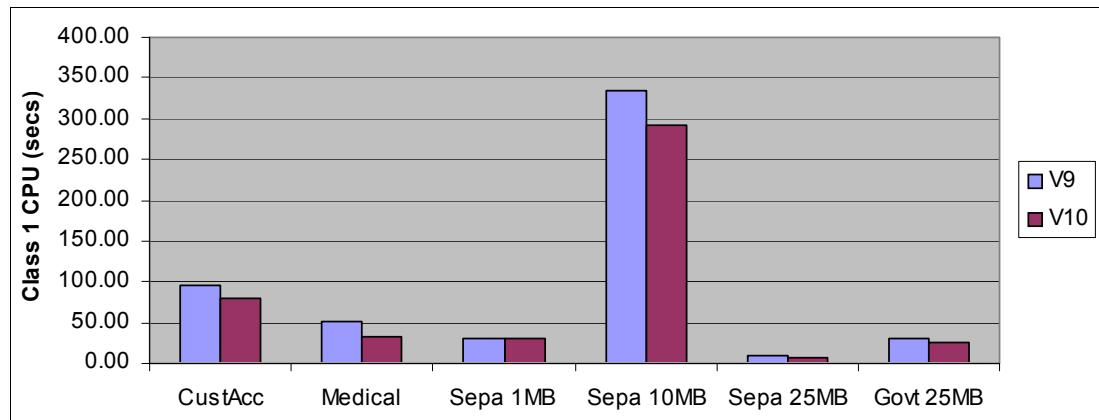


Figure 4-8 XML validation measurement results

The measurements show that the XML validation enhancement provides between a 6% and a 56% reduction in class 1 CPU time, depending on the workload and document size.

The XML schema validation enhancements in DB2 10, also available in DB2 9 with APARs PK90032 and PK90040 applied, can provide a significant reduction in class 1 CPU time for the validation of XML schemas. Since this enhancement was retrofitted to DB2 9 by maintenance, the performance benefit is greater than for the old method of using the user defined function to do the schema validation. The validation process is now also able to take advantage of available specialty engines.

The XML schema validation enhancement is available in conversion mode and in DB2 9 with the appropriate APARs applied.

4.2.5 XML type modifier

The XML data type can accept any well-formed XML documents. However, in many cases, users want to store in one XML column documents that have a similar structure or that conform to the same XML schema. DB2 10 introduces the XML type modifier which qualifies the XML data type with a set of one or more XML schemas. The value of an XML column with an XML type modifier must conform to at least one XML schema specified in the type modifier. An XML type modifier is similar to a check constraint in that it validates the data for the XML column according to a schema that is defined in the DDL.

If you are already invoking the validation function DSN_XMLVALIDATE, there is no additional cost to insert an XML document if there is an XML type modifier defined for the XML column. An INSERT with and without an XML type modifier shows the same amount of time being spent in validating documents.

The XML type modifier is available in new function mode. Multi-versioning support is not required.

4.2.6 Support for binary XML

DB2 10 introduces a binary format for XML data, which is called *Extensible Dynamic Binary XML DB2 Client/Server Binary XML Format* or XDBX. Binary XML format is an external representation of an XML value that can be used for exchange of XML data between a client and a data server. The binary representation provides efficient XML parsing, which can result in performance improvements for XML data exchange. Support of binary XML provides performance improvements, because the XML data can be encoded more efficiently.

This enhancement supports accessing binary XML data by DRDA, and for use by CLI, ODBC, JDBC, and SQLJ applications. The binary XML format can be used for INSERT and SELECT operations, as well as for the LOAD and UNLOAD utilities.

Binary XML performance

The performance evaluation of this enhancement focuses on the performance gain when binary XML is used during INSERT and LOAD processing, based on the assumption that parsing is one of the most significant factors in affecting performance.

The cost of the non validation parser process is measured as 15% to 50% of the cost of XML INSERT and LOAD processing. The objective of this enhancement is to achieve a 10% to 50% CPU reduction by using binary XML.

Performance tests

To measure the performance of binary XML, we ran the following five tests:

- ▶ Insert a 14 KB document
- ▶ Insert a 200 KB document
- ▶ Insert a 25 MB document
- ▶ LOAD utility
- ▶ UNLOAD utility

The measurements for our tests are shown in Figure 4-9. The bars labeled “XML” are for textual XML files, while the bars labeled “Binary” are for XML files converted to binary. Both sets of measurements were performed in DB2 10.

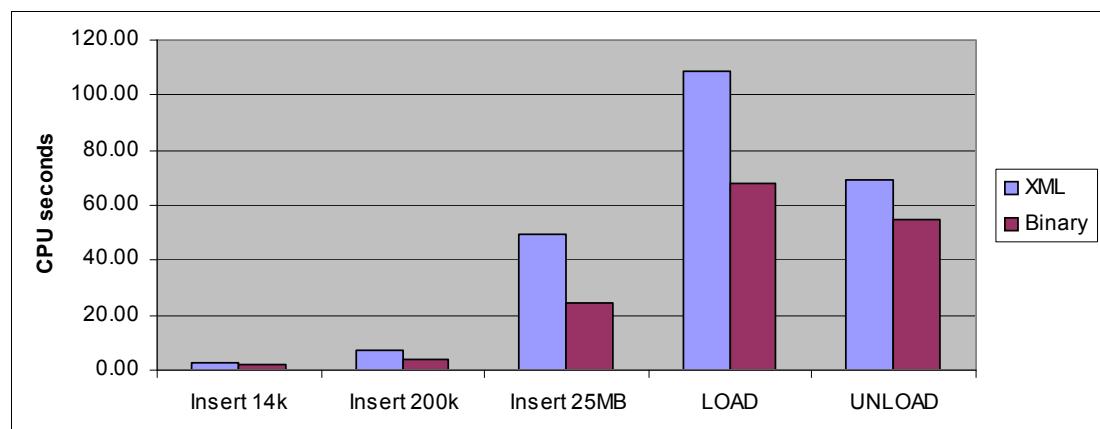


Figure 4-9 Binary XML measurement results

The measurements show that INSERT processing for binary XML documents consumed between 27% and 51% less CPU time than the same processing for textual XML documents, with greater savings as the document size increases. The LOAD utility consumed 38% less CPU time for binary XML than textual XML, while the UNLOAD utility consumed 21% less CPU time for binary than for textual.

The binary XML enhancement in DB2 10 can provide a significant reduction in CPU time for INSERT, LOAD, and UNLOAD processing. The binary representation provides efficient XML parsing, which can result in performance improvements for XML data exchange, specifically for DRDA applications and for the LOAD and UNLOAD utility.

The binary XML enhancement is available in new function mode. Multi-versioning support is not required.

4.2.7 Support for multiple versions of XML documents

In DB2 9 there are two availability or concurrency issues for XML data:

- ▶ XML locks are obtained to avoid a work file inconsistency problem that can occur for XML columns when the base table is in the work file and the XML document is deleted or updated. These locks on the work file can result in concurrency issues when updating XML documents.
- ▶ An SQL statement that modifies the value of an XML column might need to reference the version of the XML document before it is modified. To allow this self reference, the contents of the XML document are loaded into memory. DB2 system parameters XMLVALA and XMLVALS control the amount of memory for XML data. These values might need to be adjusted for updates of very large XML documents.

DB2 10 solves these issues by providing the capability to maintain multiple versions of XML documents. Old versions of updated or deleted XML documents are kept to reduce locking and improve concurrency. In addition, this versioning is required to support the new XML features described in this chapter.

Multi-versioning is not a performance feature, but it is a feature that you need in order to take advantage of the DB2 10 performance enhancement for XML described in 4.2.2, “Modifying part of an XML document”.

CPU and elapsed times for UPDATE and DELETE of multi-versioned XML tables in DB2 10 are comparable to the performance of UPDATE and DELETE for XML tables in DB2 9.

Keeping old versions of XML documents can cause XML queries to run longer because data is spread out, more I/O is required, and you can have a lower buffer pool hit ratio. Old versions of XML documents are cleaned up automatically by a background task within one to two minutes of running a massive multi-threaded update or running a mixed transaction workload.

Multi-versioning of XML documents is available in new function mode and is only available for XML columns in which the base table has been created in a universal table space (UTS) after you migrate to DB2 10 new function mode (NFM).

4.3 Inline LOBs

DB2 10 introduces a number of enhancements to the processing of large objects (LOBs). In this section we describe the performance of SQL access to inline LOBs as well as the performance of LOAD and UNLOAD of inline LOB data. The performance of using spanned records (VBS format) for inline LOB data is described in 9.8.1, “LOAD and UNLOAD with spanned records” on page 284.

For background information about LOBs, see *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270.

With DB2 10 you can additionally use the INLINE_LOB_LENGTH (maximum is 32680 bytes) subsystem parameter or the INLINE LENGTH clause with a CREATE TABLE or CREATE DISTINCT TYPE statement to specify that DB2 store a compatible portion of LOB data in the base table space with the other non-LOB data. Reordered row format is also required.

DB2 10 allows you to store LOB data using the following methods:

- ▶ In an auxiliary table, which has been the only option prior to DB2 10
- ▶ In the base table where the LOB column definition is, if the length fits within the base row, which is a new option in DB2 10 (inline LOBs)
- ▶ In the base table where the LOB column definition is, with overflow to an auxiliary table (out-of-line LOBS)

In all three alternatives, the LOB column is defined in the base table; the difference is where the LOB data is physically stored: with the base table column data, in an auxiliary table or a combination of the two. When a LOB column is defined such that all or a portion of the LOB column data is stored in a base table, then the LOB is referred to as an inline LOB. If all of the LOB column data fits in the base table, then the LOB is considered to be fully inlined.

In the following sections we discuss these topics:

- ▶ Advantages of inline LOBS
- ▶ Inline LOBs performance
- ▶ Queries for LOB size distribution
- ▶ Inline LOBs: Conclusions

4.3.1 Advantages of inline LOBS

Inline LOBs offer the following performance advantages over LOBs that are stored in auxiliary tables (sometimes called outline LOBs):

- ▶ Disk space savings because two LOBs cannot share a page on a LOB table space
- ▶ Disk space savings because the inline portion of a LOB can be compressed
- ▶ Synchronous I/Os to the AUX index and LOB table space are avoided
- ▶ CPU savings associated with accessing the AUX index and LOB table space
- ▶ Sequential and dynamic prefetch I/O for LOBs
- ▶ Improved effectiveness of FETCH CONTINUE when scanning rows
- ▶ Index on expression can be enabled for LOB data

There are a number of considerations when making the decision to store your LOB data inline or in auxiliary table spaces. We discuss the following performance considerations in the sections that follow:

- ▶ LOB column frequency of reference
- ▶ Distribution of LOB column sizes
- ▶ Page size of base table space
- ▶ Compressibility of LOB data
- ▶ Search requirements on LOB data

LOB column frequency of reference

If the LOB column is rarely referenced when data from the base table is accessed, then it does not make sense to store the LOB column data inline with the rest of the base table columns. Storing the LOB data inline will only increase the size of the base table rows, resulting in less rows stored per page and subsequently a lower hit ratio for the table as a whole.

On the other hand, if the LOB data is almost always referenced whenever the table is referenced, then it might make sense to store the LOB data inline, though the other considerations also need to be reviewed.

Distribution of LOB column sizes

If the size of the LOB column is small enough that you can store the entire LOB with the other columns of the row on the same data page, then it might make sense to inline the LOB column. When DB2 cannot fit the entire LOB in a page, then the LOB is split between the base table space and the LOB table space. The size of a LOB column can vary widely, depending on the data. The distribution of the LOB column size will impact whether you decide to inline the LOB and how much of the LOB to inline.

If the LOB column will fit entirely inline in the base table space 90% of the time, then it is clearly beneficial to inline the LOB. If the LOB column will fit entirely inline in the base table space 10% of the time, and the other 90% of the time the LOB data is split between the base table space and the LOB table space, then DB2 will need to split the LOB, resulting in extra I/O because DB2 needs to read from both the base table space and the LOB table space to return the data in 90% of the cases. In this case you might need to adjust the page size or consider not defining the LOB as an inline LOB.

Page size of base table space

The page size of the base table space and the size of the LOB column define how many rows can fit on a page. When LOB data is not stored inline, you can only store one LOB on a page of a LOB table space. With inline LOBs, you can fit as many LOBs on a page as the LOB size and the page size will allow.

If the average size of a LOB column is 3,000 bytes and the page size of the base table space is 4 KB, then you will only be able to store, on average, one row per page if you inline the LOB column, unless the LOB data compresses well. If the base table can only store one LOB per page, you might in fact see performance regression when accessing the base table due to there being so few rows per page.

Compressibility of LOB data

LOB table spaces cannot be compressed. One of the performance advantages of inline LOBs is that the inline LOB data can be compressed. LOB data that is stored in LOB table spaces cannot be compressed. Not only can compression of LOB data save you space, it can also result in reduced I/O. If a good compression ratio allows you to store more data on fewer pages, then you might be able to store a LOB column as inline and reduce the number of I/Os by eliminating the need to read a separate page for each row.

If the LOB data does not compress well, but is small in most cases, then you still might be able to reduce the number of I/Os when using inline LOBs by storing multiple LOBs on a page.

Search requirements on LOB data

Another performance advantage of inline LOBs is that you can build an index on the contents of the LOB column. DB2 10 allows you to build an index on an expression for a LOB column if it is an inline LOB. The only built-in function allowed in the expression is SUBSTR. The use of an index on expression using SUBSTR allows you to do an index search for a text string in the LOB data.

If your applications have specific search requirements for reading portions of the LOB data, and the requirements are that the data can be searched and retrieved quickly, then the LOB column might be a candidate to be defined as an inline LOB.

4.3.2 Inline LOBs performance

Taking into account the considerations just listed for determining whether or not to define a LOB as inline, we ran a number of tests to compare the performance of inline LOBs with out-of-line LOBs defined in LOB table spaces. This section describes the tests we ran and the performance measurements for each test. Unless otherwise noted, each test was run in DB2 10 comparing inline LOBs and LOBs using LOB table spaces (referred to as out-of-line or outline LOBs). Unless otherwise noted, all tests were run on a z10 processor using DS8300 DASD.

The following tests were included:

- ▶ Using inline LOBs to save DASD space
- ▶ Random access to small LOBs
- ▶ Sequential inserts with small LOBs
- ▶ Update and delete of small LOBs
- ▶ LOAD REPLACE of small LOBs
- ▶ UNLOAD of small LOBs
- ▶ Spatial support
- ▶ Inline LOB columns versus VARCHAR columns
- ▶ Buffer pool tuning for inline LOBs

Using inline LOBs to save DASD space

Inline LOBs can provide DASD space savings because two LOBs cannot share a page on a LOB table space, while two or more LOBs can share a page on table space where the LOB is defined as inline. We ran a number of tests loading 1 million LOBs of various sizes, for both inline LOBs and out-of-line LOBs, and compared the DASD space used for each case. The number of gigabytes used in each case is shown in Figure 4-10.

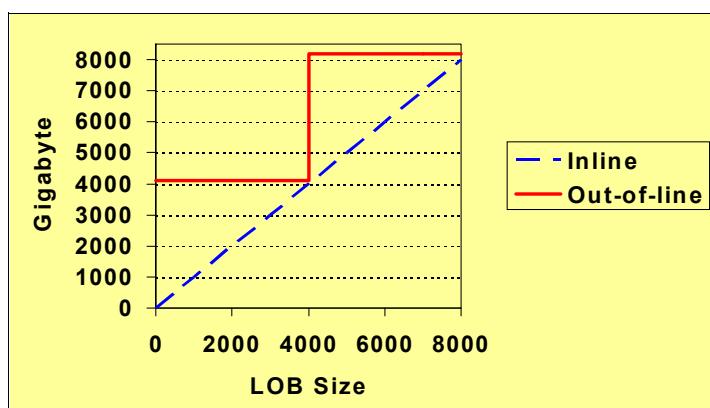


Figure 4-10 DASD space used for 1 million LOBs with 4 KB LOB page size

The graph shows that there are considerable DASD savings for LOBs whose size is not close to a multiple of 4 KB. If the LOB size is 4 KB or 8 KB, then there are no DASD savings because you can only store one LOB in a 4 KB page of a base table space. In the case of the 8 KB LOB size, half the LOB is stored in the base table space, while the other half is stored in the LOB table space.

If the LOB is much smaller than 4 KB, say 1 KB, then the DASD savings is significant. Each inline LOB consumes 1 KB of DASD space, while each LOB that is stored in a LOB table space consumes 4 KB of DASD space, or four times as much DASD. The DASD space requirements for 1 million 1 KB LOBs is 1,000 gigabytes (a terabyte) for inline LOBS and 4,000 gigabytes (four terabytes) for out-of-line LOBs.

The DASD savings for inline LOBs are greatest when you have small LOBs and many of them. Compression can also contribute to savings for inline LOBs.

Random access to small LOBs

Small LOBs are good candidates for inline LOBs because you can store many more LOBs per page, therefore reducing the elapsed time to retrieve LOB data. We ran tests to select 10,000 rows from a table that contains a LOB that is only 200 bytes long. The class 2 elapsed time to select this data for the inline and out-of-line LOB cases is shown in Figure 4-11.

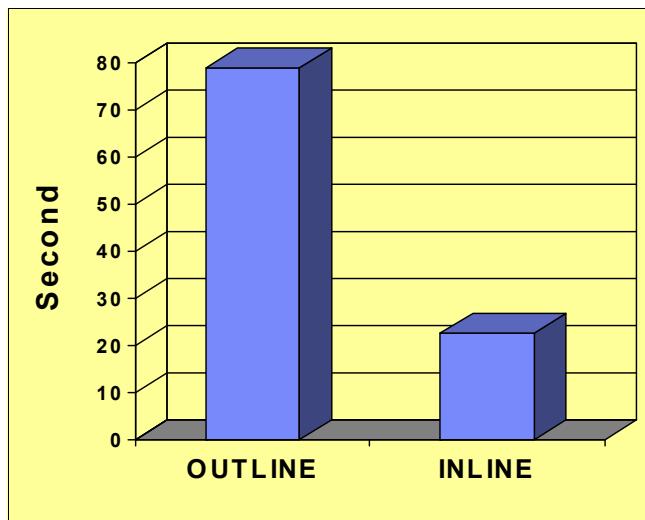


Figure 4-11 Class 2 elapsed time to select 10,000 x 200 byte LOBs

The class 2 elapsed time to select from the inline LOBs is 22.6 seconds, while the class 2 elapsed time for the out-of-line LOBs is 79.0 seconds. In this test the select of the inline LOB data was 71% faster than the same select of the same data stored in a LOB table space, due to more LOBs being stored on a page for inline LOBs and, therefore, less I/Os required to read the same amount of data.

Sequential inserts with small LOBs

Small inline LOBs also gain performance benefits when the rows are inserted, because there is no need to insert a row in the LOB table spaces or the AUX index. We ran tests to insert 10,000 rows into a table with a LOB column that is 200 bytes long. We ran the tests with an inline LOB and with an out-of-line (outline) LOB. The class 2 elapsed time to insert the rows for the inline and out-of-line LOB cases is shown in Figure 4-12.

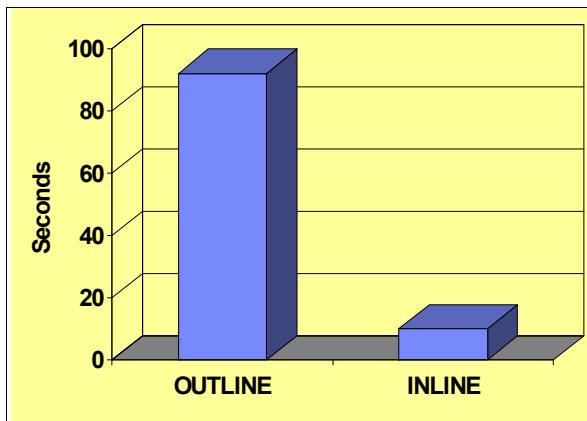


Figure 4-12 Class 2 elapsed time to insert 10,000 x 200 byte LOBs

The class 2 elapsed time to insert 10,000 inline LOBs is 10 seconds, while the class 2 elapsed time for the out-of-line LOBs is 92 seconds. In this test the insert of the inline LOB data was 89% faster than the same insert of the same data to a LOB table space, because the LOB table space and AUX index did not need to be accessed for inline LOBs.

The class 2 CPU time to insert the rows for the inline and out-of-line LOB cases is shown in Figure 4-13.

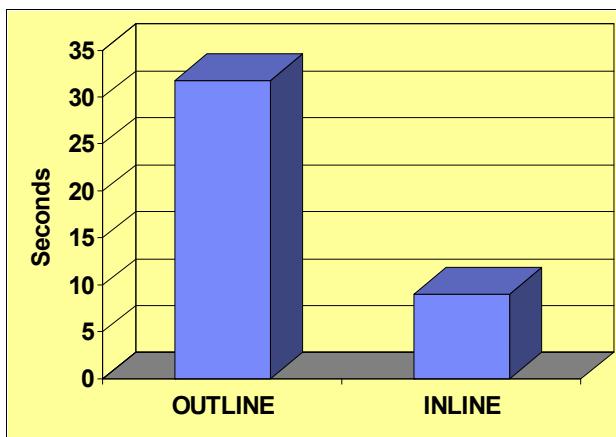


Figure 4-13 Class 2 CPU time to insert 10,000 x 200 byte LOBs

The class 2 CPU time to insert 10,000 inline LOBs is 9.017 seconds, while the class 2 CPU time for the out-of-line LOBs is 31.8 seconds. In this test the insert of the inline LOB data showed a 72% improvement in CPU time compared to the same insert of the same data to a LOB table space.

Update and delete of small LOBs

Small inline LOBs also gain performance benefits when the rows are updated or deleted. We ran tests of 5,000 random updates and deletes of rows with a LOB column that is 200 bytes long. We ran the tests with an inline LOB and with an out-of-line LOB. The class 2 elapsed time to perform the updates and deletes for the inline and out-of-line LOB cases is shown in Figure 4-14.

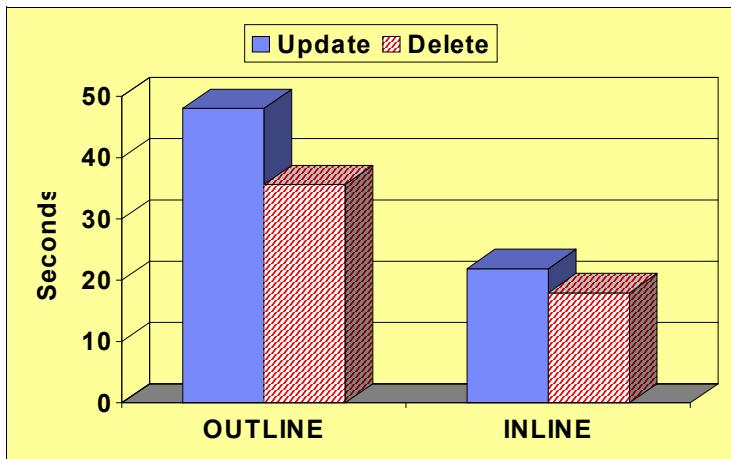


Figure 4-14 Class 2 elapsed time for 5,000 random updates/delete of 200 byte LOBs

The class 2 elapsed time to perform 5,000 random updates of the 200 byte inline LOBs is 21.899 seconds, while the class 2 elapsed time for the same updates of the out-of-line LOBs is 48.032 seconds. In this test the updates of the inline LOB data was 54% faster than the same updates of the same data to a LOB table space.

The class 2 elapsed time to perform 5,000 random deletes of the 200 byte inline LOBs is 17.992 seconds, while the class 2 elapsed time for the same deletes of the out-of-line LOBs is 35.616 seconds. In this test the deletes of the inline LOB data was 49% faster than the same deletes of the same data from a LOB table space.

The class 2 CPU time to perform the same updates and deletes of the rows for the inline and out-of-line LOB cases is shown in Figure 4-15.

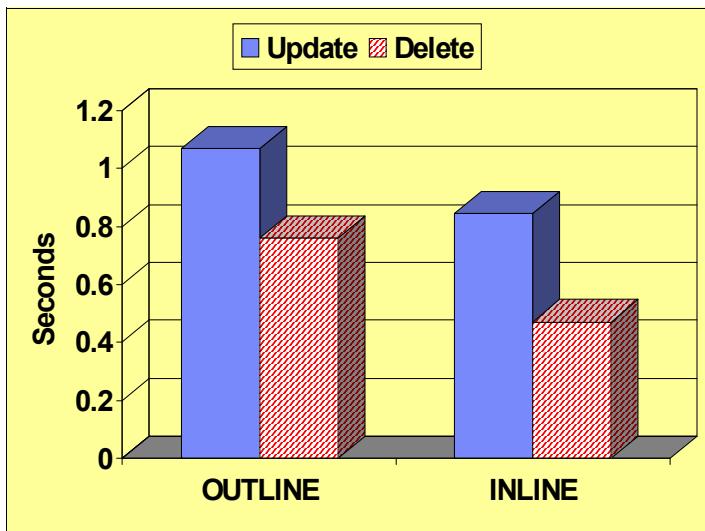


Figure 4-15 Class 2 CPU time for 5,000 random updates/delete of 200 byte LOBs

The class 2 CPU time to perform 5,000 random updates of the 200 byte inline LOBs is 0.846 seconds, while the class 2 CPU time for the same updates of the out-of-line LOBs is 1.071 seconds. In this test the updates of the inline LOB data showed a 21% improvement in CPU time compared to the same updates of the same data on a LOB table space.

The class 2 CPU time to perform 5,000 random deletes of the 200 byte inline LOBs is 0.471 seconds, while the class 2 CPU time for the same deletes of the out-of-line LOBs is 1.76 seconds. In this test the deletes of the inline LOB data showed a 38% improvement in CPU time compared to the same deletes of the same data from a LOB table space.

LOAD REPLACE of small LOBs

Small inline LOBs also gain performance benefits when the rows are loaded into the base table, because there is no need to load a row into the LOB table spaces or to create the AUX index entries. In addition, when we load small rows into inline LOB columns, we can store more than one row per page, while out-of-line LOBs can only be stored as a single LOB per page.

We ran tests to load rows with 200 byte LOBs and to load rows with 3,900 byte LOBs. We ran the tests using inline LOBs and out-of-line LOBs. For the 200 byte test cases we can store many rows in a page when we use inline LOBs, while we can only store one row per page for the out-of-line LOBs. For the 3,900 byte test cases we can only store a single row per page, regardless of whether we use inline or out-of-line LOBs, so we do not expect to see as great a savings.

The class 1 elapsed time to load the 200 byte and the 3,900 byte LOBs for the inline and out-of-line LOB cases is shown in Figure 4-16.

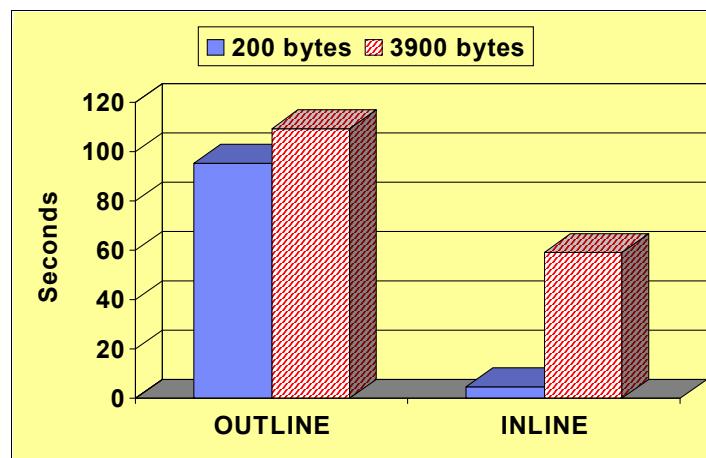


Figure 4-16 Class 1 elapsed time for LOAD REPLACE of small LOBs

Notice the difference in elapsed time savings when loading a small LOB (200 bytes) versus a larger LOB (3,900 bytes). For the 3,900 byte LOB, we have a 46% reduction in class 1 elapsed time (from 109.5 seconds to 59.0 seconds) when we switch from out-of-line LOBs to inline LOBs. For the 200 byte LOB, we have a 95% reduction in class 1 elapsed time (from 95.4 seconds to 4.64 seconds) when we switch from out-of-line LOBs to inline LOBs. These numbers show the advantages of storing smaller LOBs inline.

The CPU savings are similar, as shown in Figure 4-17.

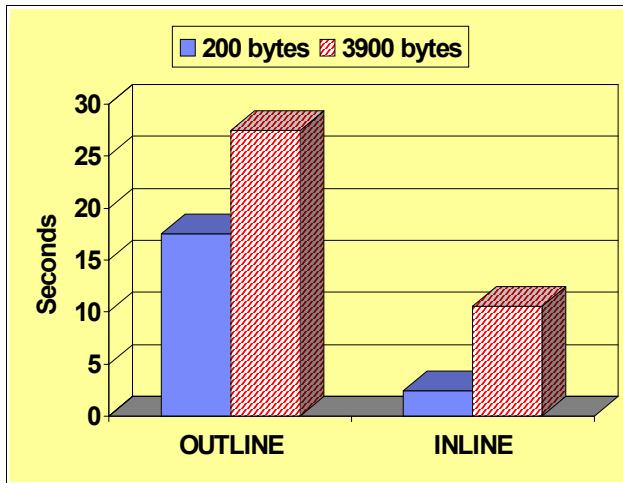


Figure 4-17 Class 1 CPU time for LOAD REPLACE of small LOBs

Notice the difference in CPU time savings when loading a small LOB (200 bytes) versus a larger LOB (3,900 bytes). For the 3,900 byte LOB, we have a 62% reduction in class 1 CPU time (from 27.5 seconds to 10.585 seconds) when we switch from out-of-line LOBs to inline LOBs. For the 200 byte LOB, we have a 86% reduction in class 1 CPU time (from 17.5 seconds to 2.4 seconds) when we switch from out-of-line LOBs to inline LOBs. These numbers also show the advantages of storing smaller LOBs inline.

UNLOAD of small LOBs

We ran the same tests we used to load the data to unload the data as well. The class 1 elapsed time to unload the 200 byte and the 3,900 byte LOBs for the inline and out-of-line LOB cases is shown in Figure 4-18.

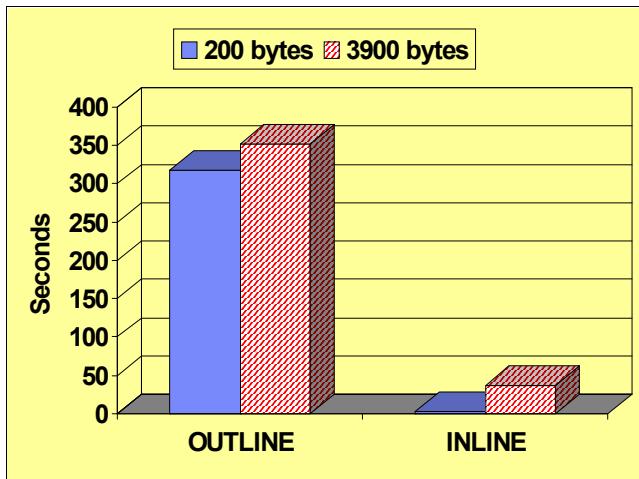


Figure 4-18 Class 1 elapsed time for UNLOAD of small LOBs

For unload, the elapsed time savings are similar for small and larger LOBs. For the 3,900 byte LOB, we have a 90% reduction in class 1 elapsed time (from 351.9 seconds to 36.55 seconds) when we switch from out-of-line LOBs to inline LOBs. For the 200 byte LOB, we have a 99% reduction in class 1 elapsed time (from 318.4 seconds to 2.529 seconds) when we switch from out-of-line LOBs to inline LOBs.

The CPU savings for the unload test cases are shown in Figure 4-19.

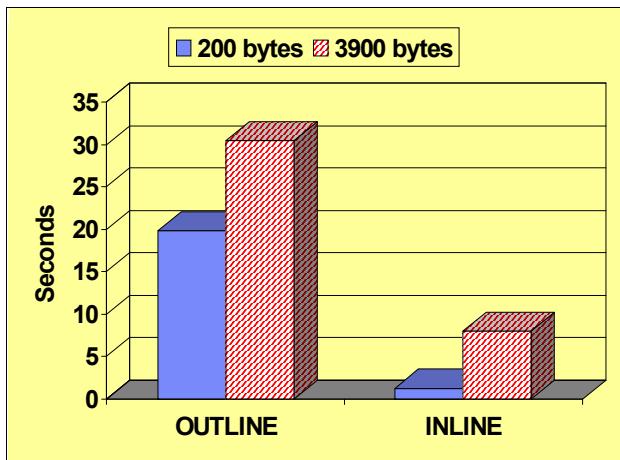


Figure 4-19 Class 1 CPU time for UNLOAD of small LOBs

For the 3,900 byte LOB, we have a 74% reduction in class 1 CPU time (from 30.46 seconds to 7.96 seconds) when we switch from out-of-line LOBs to inline LOBs. For the 200 byte LOB, we have a 94% reduction in class 1 CPU time (from 19.8 seconds to 1.287 seconds) when we switch from out-of-line LOBs to inline LOBs. These numbers show the advantages of storing smaller LOBs inline.

Spatial support

Spatial support was introduced in DB2 9 to manage geospatial information. Spatial queries use a type of index called a spatial grid index. The indexing technology in IBM Spatial Support for DB2 for z/OS utilizes grid indexing, which is designed to index multidimensional spatial data, to index spatial columns. IBM Spatial Support for DB2 for z/OS provides a grid index that is optimized for two-dimensional data on a flat projection of the Earth.

DB2 10 improves the performance of spatial support by exploiting the capability to create a spatial grid index on inline LOB columns.

To test the performance advantages of inline LOBs for spatial support, we ran a test creating a spatial index in DB2 9 and creating a spatial index on an inline LOB in DB2 10. The class 1 elapsed times for each test is shown in Figure 4-20.

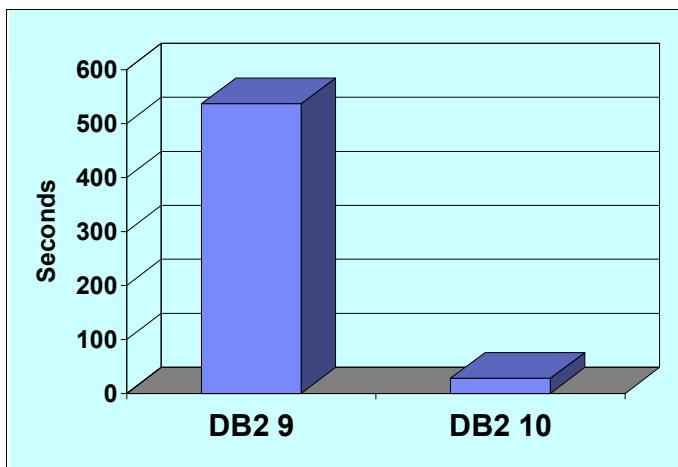


Figure 4-20 Class 1 elapsed time for create spatial index on inline LOB column

In our tests the class 1 elapsed time to create the spatial index on the LOB column in DB2 9 was 536.26 seconds. The class 1 elapsed time to create the spatial index on the inline LOB column in DB2 10 was 27.71 seconds. This is a savings of 95%.

After the spatial index was created in each version, we ran a series of spatial queries to test the performance of the spatial index for the inline LOB in DB2 10. Figure 4-21 shows the class 1 elapsed time measurements for four spatial queries run in DB2 9 with a spatial index on the out-of-line LOB column and for the same four queries run in DB2 10 with a spatial index on the inline LOB column.

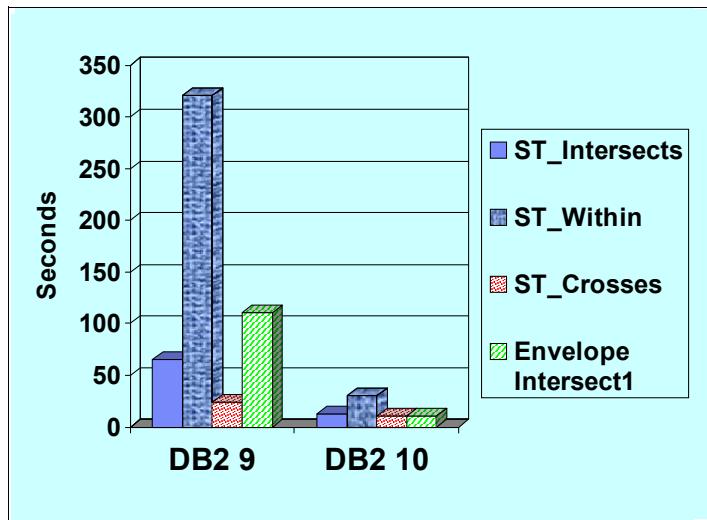


Figure 4-21 Class 1 elapsed time for spatial queries using spatial indexes

For all four types of queries we see class 1 elapsed time savings using the index on the inline column in DB2 10. The savings are as follows:

- ▶ ST Intersects query - 80% (13 seconds compared to 65 seconds)
- ▶ ST_Within query - 91% (30 seconds compared to 321 seconds)
- ▶ ST_Crosses - 45% (11 seconds compared to 24 seconds)
- ▶ Envelope_Intersect1 - 90% (11 seconds compared to 111 seconds)

As shown before, the ability to create a spatial index on inline LOB columns in DB2 10 provides significant savings for many types of spatial queries.

Inline LOB columns versus VARCHAR columns

If 100% of your LOBs will be inlined, then you can use VARCHAR instead. However, it only takes one row to exceed the VARCHAR limit that necessitates the need to use a LOB. The performance of fully inlined LOBs is very similar to VARCHAR. If we assume that a ROWID column is defined, with or without a LOB column, the CPU time for most operations that use an inline LOB column is within 10% of the CPU time for an equivalent VARCHAR column.

Buffer pool tuning for inline LOBs

The size of the LOB column and the page size of the base table space are two factors that can impact the performance of inline LOBs. Let us look at some test cases involving different distributions of LOB sizes and discuss some buffer pool tuning options to most efficiently access inline LOBs without impacting access to other objects. These test cases demonstrate some rules of thumb that you can use to tune your LOB data and associated buffer pools. You need to also take into account other factors, such as buffer pool hit ratios and DASD I/O performance, when tuning LOB buffer pools. In other words, you need to apply the same principles that you might apply for buffer pool tuning in general.

Figure 4-22 shows four different hypothetical LOB size distributions. For these four test cases, 96% to 99% of the LOBs are less than 32,000 bytes. If we want to increase the page size of the base table space to 32 KB, we can potentially inline 99% of the LOBs in case 1, and 96% of the LOBs in case 4.

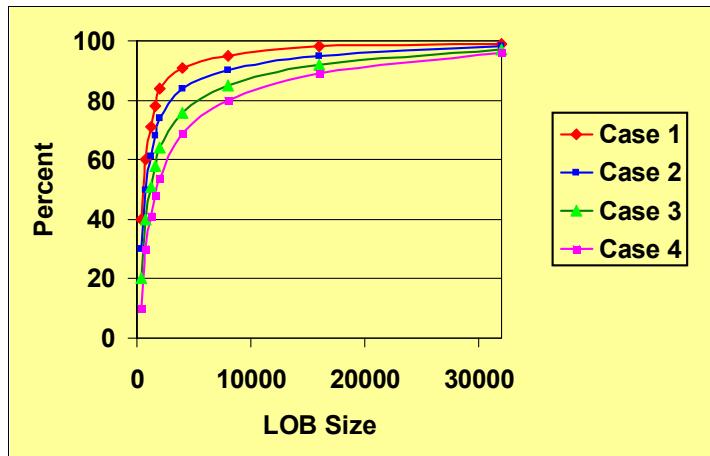


Figure 4-22 Tuning inline LOBs - Cumulative LOB size distribution - test cases 1 to 4

In case 1, if we stay with a 4 KB page, then we can inline 91% of the LOBs. The other 9% of rows will “pollute” the base table space buffer pool because only one row will fit on a page and the LOB will be split between the base table space and the LOB table space. When we split a LOB, we introduce the possibility of an additional synchronous I/O to the base table space (and we still have to probe the AUX index and access the LOB table space). However, for the other 91% of the rows, inlining the LOBs will save a lot more I/Os than 9%. So, we save a lot by using inline LOBs.

When choosing the page size for a table space that includes inline LOBs, we have to look at what the difference is in the percent of LOB values that can be fully inlined as we increase the page size. For example, for test case 1, a 4 KB page size allows us to fully inline 91% of the LOBs. Increasing the page size to 8 KB will allow us to fully inline 95% of the LOBs, increasing the page size to 16 KB will allow us to fully inline 98% of the LOBs and increasing the page size to 32 KB will allow us to fully inline 99% of the LOBs for test case 1. The performance advantages of being able to fully inline 99% of your LOBs is that you rarely ever (only 1% of the time) have to probe the AUX index and access the LOB table space. For test cases 1 through 4 you will get greater performance with a 32 KB page size and the maximum inline quantity.

Now let us look at four cases where the LOB sizes are larger. Figure 4-23 shows four more hypothetical LOB size distributions. In each of the four test cases, there will be much more pollution of the base table space than what we saw for test cases 1 through 4.

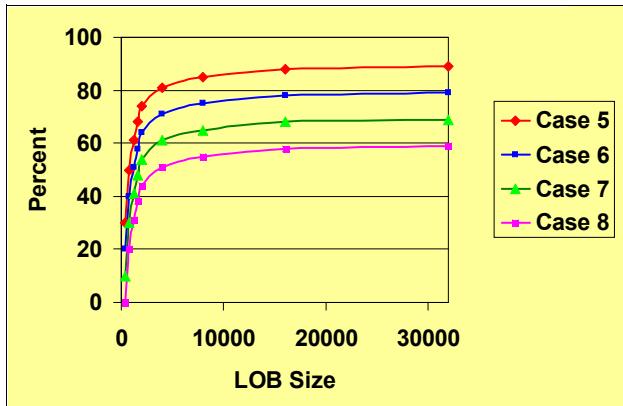


Figure 4-23 Tuning inline LOBs - Cumulative LOB size distribution - test cases 5 to 8

For test case 5, we can fully inline only 81% of the rows if we choose a 4 KB page size. However, even if we choose a 32 KB page size we can only fully inline 89% of the rows, which means that 11% of the LOBs will pollute the base table space and be split between the base table space and the LOB table space. Increasing the percentage we can fully inline from 91% to 99%, such as in test case 1, provides us with tremendous performance benefits. Increasing the percentage we can fully inline from 81% to 89% is not so dramatic an improvement because we still have to read from the LOB table space 11% of the time.

For these test cases, it makes more sense to choose a 4 KB page size and specify the maximum inline quantity that is suitable for a 4 KB page. In addition, for test cases like these you might want to consider whether or not you want to inline the LOBs at all. If you were getting a good hit ratio on the table before inlining the LOB, then inlining it might hurt the hit ratio and negatively impact performance.

When managing the objects for LOBs, separating indexes, tables and LOBs into different buffer pools is a common practice. This is done to prevent tables from polluting the buffer pool used for indexes, and to prevent LOBs from polluting the buffer pool used for data. By dividing the buffer pools in this manner, you can assign a small buffer pool to LOB table spaces and dedicate more storage to your indexes and data. This strategy makes sense if a larger LOB buffer pool is unlikely to improve the LOB buffer hit ratio.

Row size can have a big effect on the buffer hit ratio. If every row has an equal probability of being referenced, then a page that has multiple small rows has a higher probability of being re-referenced than a page that has a small number of large rows. It is not practical or effective to chop up the buffer pool space into too many buffer pools based on row size, but it is reasonable to create one small buffer pool for the sacrificial lambs, so to speak. This is the buffer pool to use when it is hopeless to expect a page re-reference. Generally speaking, this is the buffer pool that needs to include LOB table spaces. This will prevent the hopeless objects from polluting the buffer pools that are being used by objects which have a good chance of achieving buffer hits. Only if you are able to achieve very high hit ratios for the base table space, then you need to consider adding more storage for the LOB buffer pool.

LOBs, at least the portion that cannot be inlined, tend to be “buffer hostile” because they eat up so much of the buffer pool that there is less available for other objects. Therefore LOB table spaces for larger LOBs that cannot be fully inlined need to be isolated in separate smaller buffer pools, allocating just enough storage to enable prefetch and deferred writes to perform well.

Indexes tend to be the most “buffer friendly” objects. therefore the index buffer pools must be made just large enough to avoid index I/Os.

If the base tables are “buffer neutral”, meaning they are not likely to dominate buffer pools, yet are also not likely to easily be stored entirely in memory, then the remainder of your available buffer pool storage has to be used for tables. Only if the tables can be made “buffer friendly,” then you need to allocate more than the minimum buffer pool sizes to LOB table spaces.

How do we handle the cases where the tables are not “buffer hostile”, but the LOBs are? In these cases we want to avoid inlining LOBs if more than half of the LOBs do *not* fit in a 4 KB page. If more than half of the LOBs fit in a 4 KB page, then we need to consider how many more LOBs will fit inline as we increase the size of the page incrementally. If an incremental increase in page size enables 10% more LOBs to fit inline, then it makes sense to increase the page size.

Let us look at one more set of test cases to demonstrate this point. Figure 4-24 shows three test cases where the LOB size distribution is such that fewer LOBs will fit in a 4 KB page than for test cases 1 through 8.

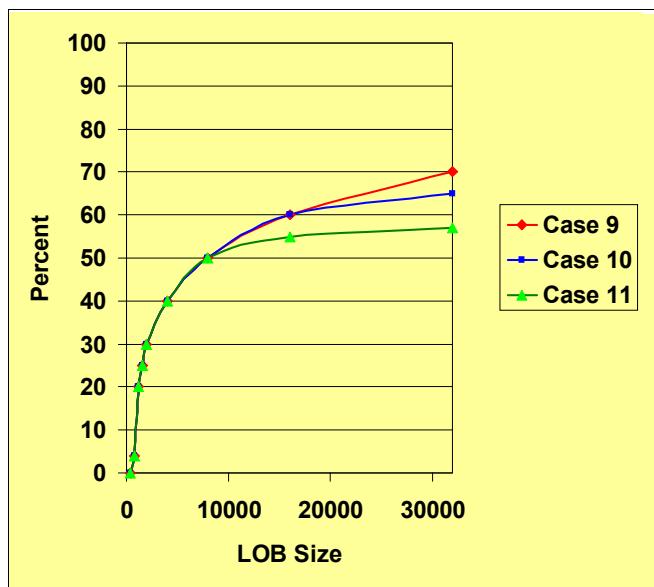


Figure 4-24 Tuning inline LOBs - Cumulative LOB size distribution - test cases 9 to 11

The LOB size distribution numbers that represent the points on the graph are shown in Table 4-1.

Table 4-1 Cumulative LOB size % distribution - test cases 9 to 11

LOB size	400	800	1200	1600	2000	4000	8000	16000	32000
Case 9	0	4	20	25	30	40	50	60	70
Case 10	0	4	20	25	30	40	50	60	65
Case 11	0	4	20	25	30	40	50	55	57

For test case 9, as we increase the page size from 16 KB to 32 KB, we are able to inline 10% more LOBs (70% compared to 60%), so we must use a page size of 32 KB for test case 9.

For test case 10 as we increase the page size from 16 KB to 32 KB we will only be able to inline 5% more LOBs (65% compared to 60%). We see a 10% increase (60% compared to 50%) when we increase the page size from 8 KB to 16 KB. Therefore we need to use a page size of 16 KB for test case 10.

For test case 11 we only see a 2% increase in the number of LOBs we can inline when we go from a 16 KB to a 32 KB page size. We only see a 5% increase when we go from an 8 KB page size to a 16 KB page size. We do see a 10% increase when we go from a 4 KB page size to an 8 KB page size. Therefore we need to use a page size of 8 KB for test case 11.

4.3.3 Queries for LOB size distribution

Statistics on the distribution of LOB column lengths are not maintained in the DB2 catalog. If you want to produce a LOB column size distribution to help you determine the appropriate inline length and page size for your LOB data, you can run a query to produce this data. A sample query to produce a LOB column size distribution is shown in Example 4-6.

Example 4-6 Sample query to produce LOB column size distribution

```
WITH LOB_DIST_TABLE (LOB_LENGTH, LOB_COUNT) AS
(
  SELECT LOBCOL_LENGTH, COUNT(*)
  FROM
  (
    SELECT ((LENGTH(STATEMENT) / 4000) + 1) * 4000
      AS LOBCOL_LENGTH
    FROM SYSIBM.SYSPACKSTMT
  ) LOB_COL_LENGTH_TABLE
  GROUP BY LOBCOL_LENGTH
)
SELECT '04000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 4000
UNION
SELECT '08000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 8000
UNION
SELECT '12000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 12000
UNION
SELECT '16000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 16000
UNION
SELECT '20000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 20000
UNION
SELECT '24000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 24000
UNION
SELECT '28000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 28000
UNION
SELECT '32000', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 32000
UNION
SELECT '99999', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
```

```
WHERE LOB_LENGTH <= 99999  
;
```

This sample query extracts column length information from the column STATEMENT in DB2 catalog table SYSIBM.SYSPACKSTMT. That column is defined with a data type of CLOB. The LENGTH function pulls the actual length of the column for each row.

- ▶ Dividing by 4000 reduces it to a multiple of 4000, stripping trailing digits from the length.
- ▶ Adding 1 to the result then bumps the number up to the next whole multiple of 4000.
- ▶ Multiplying by 4000 returns the result to an actual value that is a multiple of 4000.

The end result is a report that shows how many LOB rows fit into each multiple of 4000 bytes in length. The counts in the second column are a cumulative distribution. When we tested the query on our DB2 10 system, we produced the report shown in Example 4-7.

Example 4-7 Report produced from LOB column size cumulative distribution query

LOB_LENGTH	LOB_COUNT
04000	35208
08000	35237
12000	35251
16000	35255
20000	35263
24000	35263
28000	35263
32000	35263
99999	35263

```
DSNE610I NUMBER OF ROWS DISPLAYED IS 9  
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
```

You can run a similar query on your existing LOB columns to see whether or not it will be worthwhile to inline those LOB columns and, if so, what page size and inline length you must use. You just need to replace SYSIBM.SYSPACKSTMT with the name of your table that contains the LOB column and replace STATEMENT with the name of the LOB column. You can also change the divider and multiplier factor from 4000 to a smaller number if you wanted to see distribution in groups of a smaller multiple instead of 4000s. For example, the results in Example 4-7 show that the vast majority of the LOB columns (35,208 of 35,263) are 4,000 bytes or less.

The LOB column size distribution will impact the maximum inline length you choose. The considerations for choosing maximum inline length are the same as for choosing the maximum length for VARCHAR columns. You need to consider the size of the row, the size of the page and the ability to alter the row to add columns while still fitting the row on the existing page size.

The query in Example 4-8 can help you build the distribution query in Example 4-6 for all LOBs created by tables with a specific TBCREATOR.

Example 4-8 Building the query in Example 4-6 for all LOBs in tables created by a TBCREATOR

```
SELECT  
'WITH LOB_DIST_TABLE (LOB_LENGTH, LOB_COUNT) AS  
(  
SELECT LOBCOL_LENGTH, COUNT(*) FROM (SELECT ((LENGTH('||STRIP(NAME)||') / 4000) + 1) * 4000 AS LOBCOL_LENGTH FROM '  
||STRIP(TBCREATOR)||'.'||STRIP(TBNAME)||'  
) LOB_DIST_TABLE  
GROUP BY LOBCOL_LENGTH  
)
```

```

SELECT ''04000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 4000
UNION
SELECT ''08000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 8000
UNION
SELECT ''12000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 12000
UNION
SELECT ''16000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 16000
UNION
SELECT ''20000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 20000
UNION
SELECT ''24000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 24000
UNION
SELECT ''28000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 28000
UNION
SELECT ''32000'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 32000
UNION
SELECT ''99999'', SUM(LOB_COUNT)
FROM LOB_DIST_TABLE
WHERE LOB_LENGTH <= 99999;
FROM SYSIBM.SYSCOLUMNS
WHERE TBCREATOR LIKE 'WDA%' AND
COLTYPE IN ('BLOB','CLOB','DBCLOB') AND
TBNAME NOT IN (
SELECT NAME FROM SYSIBM.SYSTABLES WHERE TYPE='X');
TERMINATE;

```

If you want to find out how many inline LOBs you have in a specific database or schema, you can issue an SQL statement like:

```
SELECT * FROM SYSIBM.SYSCOLUMNS WHERE TBCREATOR LIKE 'SYS%' AND COLTYPE IN ('BLOB','CLOB','DBCLOB') AND LENGTH > 4;
```

4.3.4 Inline LOBs: Conclusions

Inline LOBs can provide significant CPU savings by reducing the need to access the AUX index and the LOB table space for LOB data. Inline LOBs are especially useful for smaller LOBs where the vast majority of the time the LOB data can be stored with the other columns of the row on the same data page of the base table.

You need to review the following considerations when deciding whether or not to inline your LOB data and how much of the data to inline:

- ▶ LOB column frequency of reference
- ▶ Distribution of LOB column sizes
- ▶ Page size of base table space
- ▶ Compressibility of LOB data
- ▶ Search requirements on LOB data

In addition, you will need to consider the above factors when choosing between defining your data as a LOB or as a VARCHAR column.

You can run a query to produce a LOB column size distribution to help you to make the right decision.

The ability to store LOB data inline in the base table is available in new function mode.

Considerations for maintenance: APAR PM29037 provides the function to ALTER the inline length for a LOB and materialize the change by REORG SHRLEVEL CHANGE.

4.4 Hash access

DB2 10 introduces an access type called hash access, which is supported by a hash space. Hash space applies to an underlying UTS table space, either PBG or PBR. Reordered row format is also required. DB2 uses a proprietary hash algorithm on the hash space to determine the location within the table space where to store and fetch the data rows. A hash space does not use a cluster index. In most cases, this direct hash access reduces data access to a single getpage, decreasing the CPU consumption and reducing application response time. Queries that use full key equal predicates on large tables, such as customer number or product number lookups, are good candidates for hash access. You can still create indexes to support other range, list, or keyed access types as you might if you used a traditional clustering index.

There are certain trade-offs for using hash access:

- ▶ A hash table is expected to consume between 1.2 and 1.5 times the disk space to obtain almost single page access avoiding rows to be reallocated on overflow area.
- ▶ MEMBER CLUSTER, table APPEND option, and clustering indexes cannot be used because the hash determines the placement of the rows. Any performance benefits from these methods are not applicable.
- ▶ Update of a hash key is not allowed because it requires the row to move physically from one location to another. This move causes DB2 semantic issues, because you can see the same row twice or miss it completely. (The same issue existed with updating partitioning keys.) DB2 returns -151 SQLCODE and 42808 SQLSTATE. The row must be deleted and reinserted with the new key value.
- ▶ Query parallelism is not available.

4.4.1 Choosing hash table candidates

Hash tables are not a good choice for all applications and must be viewed as yet another design tool in the DBA toolbox. Use hash tables selectively to solve specific design problems. In fact, a poor choice for hash access can severely impact query and insert performance through *death by random I/O* because there is no concept of clustering.

Evaluate the applications and workload thoroughly before adopting hash organization. Hash organized tables deliver the most reductions and response time improvements in certain specific situations, such as:

- ▶ The table has a unique key
- ▶ Queries that specify equality predicates on unique values to return a single row of data or columns from a single index key with accessing the row.
- ▶ Most access to the data in the table is truly random.

Applications that use range scans based on a cluster index might not perform optimally with hash organized tables. You can use IFCID199 to verify that access is truly random.
- ▶ The size of the data in the table is relatively stable, or the maximum size of the data is known. The amount of space that must be dedicated to a hash organized table is fixed. When the size is known, the correct HASH SPACE can be chosen at CREATE TABLE time preventing overflow area usage as more and more rows are inserted.
- ▶ Many rows fit on a single data page.

When too few rows fit within a single data page, additional space might be required to achieve the benefits of hash organization.
- ▶ The table contains rows of relatively uniform size.
- ▶ The benefits of hash access increase as the number of levels of an index on the table's unique key increases beyond 2. Hash access is significantly better when replacing indexes that are 3 levels or more deep.

After adopting hash organization, monitor real time statistics to ensure that hash access is used, and tune the size of the hash space.

You can either create tables for hash access or alter existing universal table spaces, by specifying the ORGANIZE BY HASH clause on the CREATE TABLE and ALTER TABLE statements.

A hash table can exist only in a universal table space, either a partition-by-growth or partition-by-range table space.

- ▶ For a range-partitioned table space, the partitioning columns and hashed columns must be the same; however, you can specify extra columns in the hash key. DB2 determines how many pages are within each partition and allocates rows to partitions based on the partitioning key. Row location within the partition is determined by the hash algorithm.
- ▶ For a partition-by-growth table space, DB2 determines how many partitions and pages are needed. Rows are inserted or loaded in the partition and page according to the hash algorithm. Rows are, therefore, scattered throughout the entire table space.

See *DB2 10 for z/OS Technical Overview*, SG24-7892, for details on hash definition.

4.4.2 Hash overflow area

DB2 appends a hash overflow area to the hash table, and a corresponding overflow index. If an insert hashes to a full page in the hash table, the new row is stored in the overflow area. The bigger the hash table is, the less likely it is that the overflow area will be used (which is good for performance), but that also means that more disk space will be used.

Hash accesses to the hash area incur only 1 getpage, but if hash overflow accesses are required then a minimum of 3 getpages (1 hash target page + 1 index leaf page + 1 overflow page) are executed, depending on the number of levels for the overflow index.

See Figure 4-25 for an illustration of these concepts.

Fortunately, in real life, the overflow index is small and is probably in the buffer pool. The probability of a row ending in the hash overflow area will depend upon the row size and adequate hash area size, and generally assumes a normal distribution.

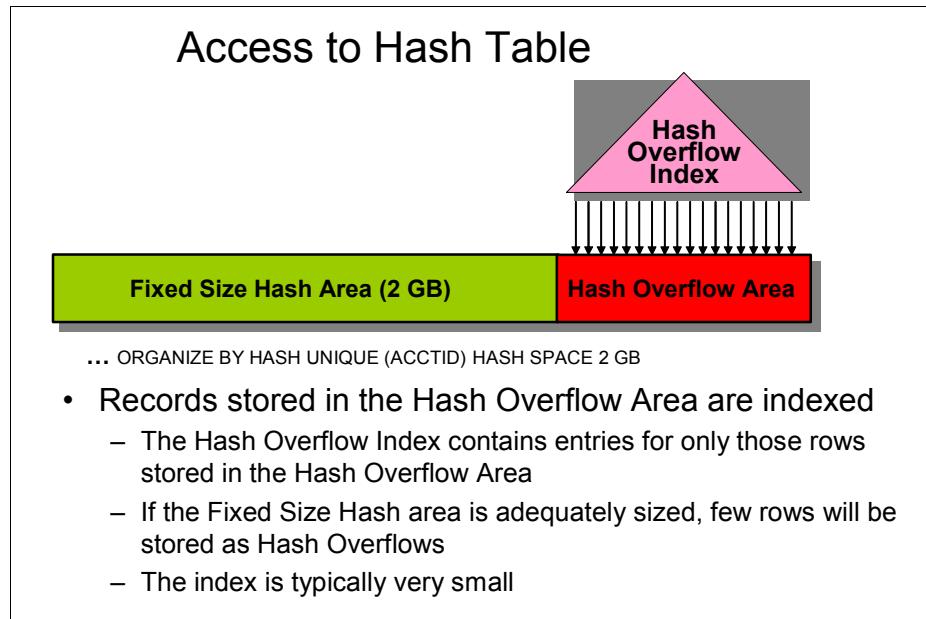


Figure 4-25 Access to hash table

Here is an example of statistical normal distribution as implemented by the hashing function on a laboratory prototype for 20 million rows over 1 million pages for a table that has 20 rows per page on average:

- ▶ Only 9% of pages get exactly 20 rows.
- ▶ Over 4% get 25; it means 5 were overflows.
- ▶ About 56% of pages have 20 or fewer entries.
- ▶ About 8.9% of rows overflow is stored in the Hash Overflow Area.

How can you reduce the percentage of overflows in a hashed table?

REORG AUTOESTSPACE(YES) already tries to adjust space so that 5 to 10% of rows overflow. If you want fewer pages allocated in the overflow area, you can increase the fixed size hash area. In our prototype, with a 20% larger hash area, 83% of pages have 20 entries or less and only about 3.1% of rows overflow.

In a traditional table space, if a variable length (or compressed) row is updated and can no longer fit in the original page, DB2 searches for a new page and creates an *indirect reference*, as noted by NEARINDREF and FARINDREF statistics in SYSIBM.SYSTABLEPART. In this case, the original row location is updated to point to the new location. Hash tables can contain indirect references too. DB2 uses the overflow area for indirect references, but the overflow index is not used.

4.4.3 Converting to hash tables

If you decide to implement hash access for some tables, Figure 4-26 shows the overview process of converting or creating a hash table.

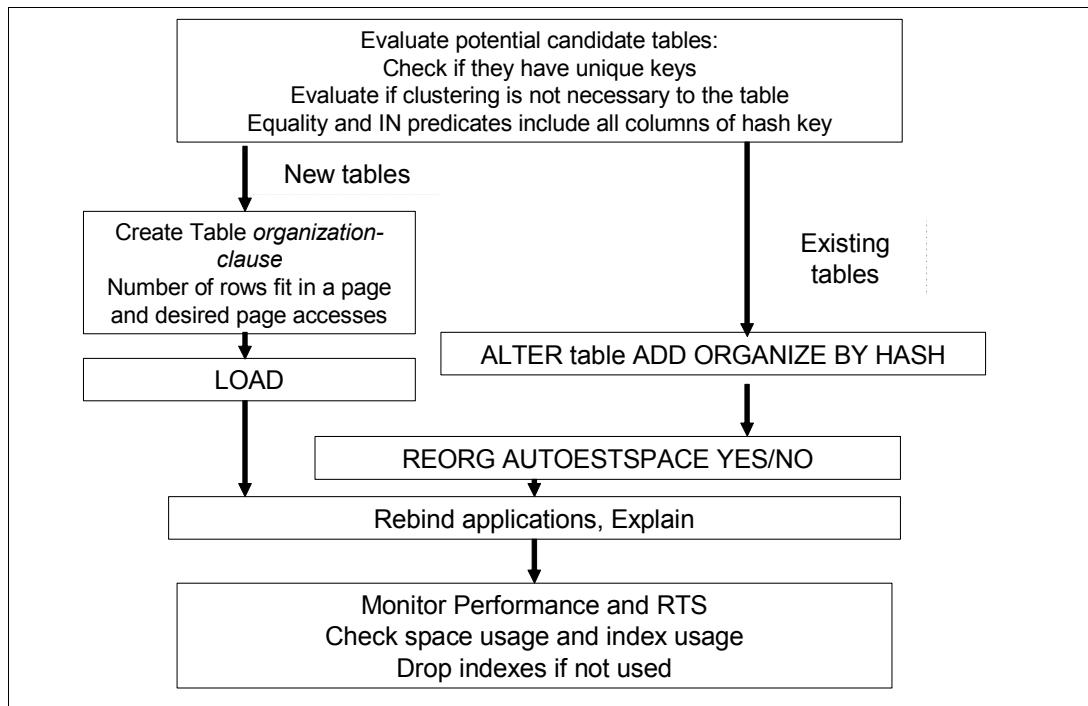


Figure 4-26 Hash table - Overview process of converting and creating

The process for converting an existing UTS table space from index access to hash access need to include the following considerations:

1. Select the candidates and check that the key conditions are met.
 - Is this access random?
 - Is this search always a fully qualified key equality (equals or IN)?
 - Is the process skip sequential? Then the random organization can provide worse performance.
 - Is cluster needed? If yes, then this is not a good candidate.
 - Can you allow more space?
2. Migrate to hash
 - ALTER and REORG the table space
 - REBIND the applications
3. Make sure that the benefits are delivered
 - Checking that hash access is used
 - Checking If you are getting many rows overflow, then you might need to increase space
 - Are the performance improvements provided?
4. At some point in the future, check for indexes that are not used and can be dropped

4.4.4 SQL access performance

The hash access is chosen as the access path when the SELECT statement includes values for all those columns in the key. These values also need to be compared using an equals predicate or through an IN list. You can check whether hash access is chosen by reviewing the PLAN_TABLE for the new values in ACESSTYPE. When ACESSTYPE is 'H', or 'HN', or 'MH', the ACCESSNAME contains the name of the hash overflow index. Hash access can also be chosen for access to the inner table of a nested loop join. However, hash access is not used in star joins.

DB2 follows the normal locking scheme based on the applications or SQL statement's isolation level and the LOCKSIZE definition of the table space. However, a new lock type of hash value is introduced to serialize hash collision chain updates when the page latch is not sufficient as in the case when the hash collision chain continues to the hash overflow area.

In our tests, we measured the performance of various accesses to a hash table.

Select hash table versus select 3-level indexed table

Figure 4-27 shows a Class 2 CPU reduction of 13% for selecting 1 row per commit and a 37% reduction for selecting 50,000 rows per commit when comparing the hash table against a 3-level indexed access and table access. I/Os are executed to both data and index leaf pages.

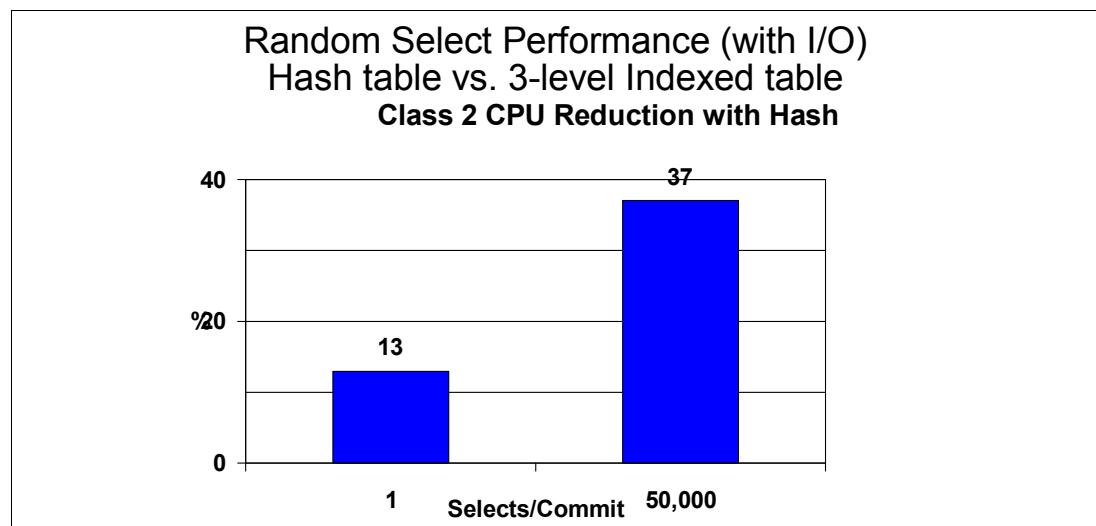


Figure 4-27 Select hash table versus Select 3-level indexed table

Select hash table versus select 3-level indexed access only

In this test we compare hash access to index-only access. Figure 4-28 shows a Class 2 CPU reduction of 9% for selecting 1 row per commit when comparing hash table against a 3-level index-only access. In this case there is no I/O to data nor index leaf pages.

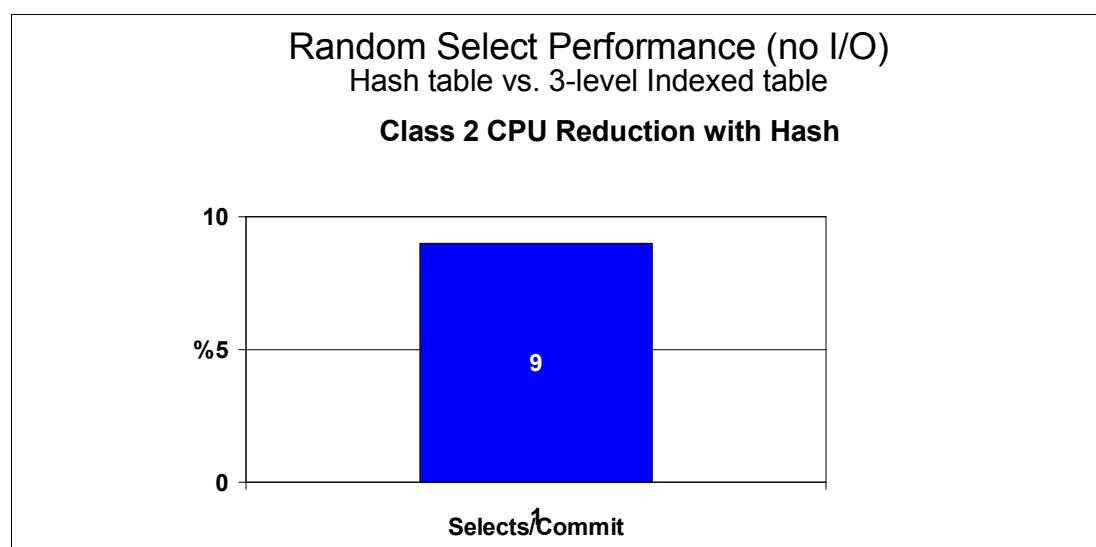


Figure 4-28 Select hash table versus Select 3-level indexed access only

Update hash table versus update 3-level indexed table

Figure 4-29 shows a Class 2 CPU reduction of 8% for updating 1 row per commit and 22% reduction for updating 50,000 rows per commit when comparing hash table against a 3-level indexed table.

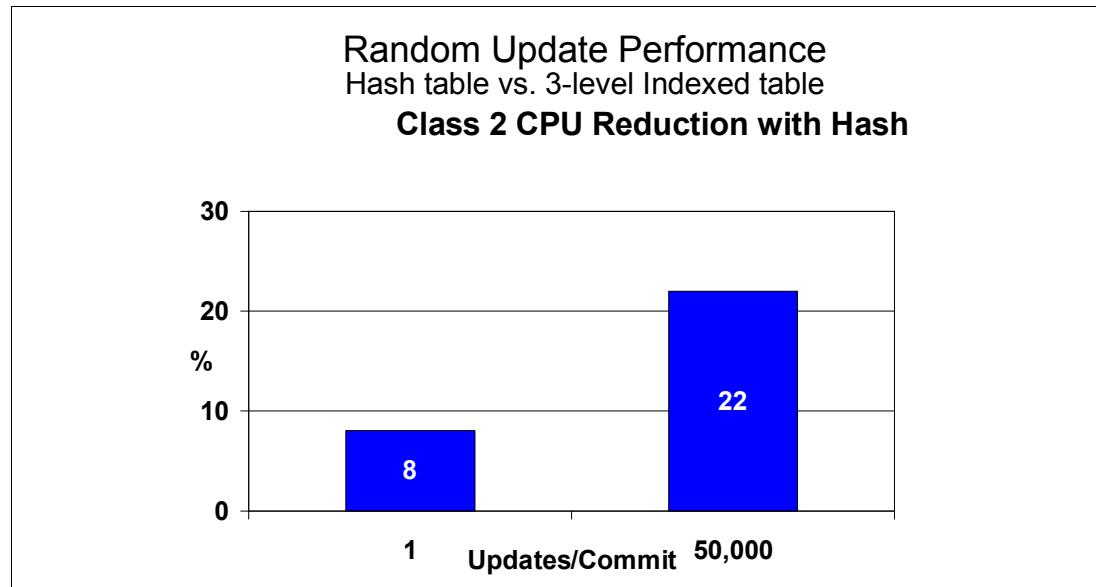


Figure 4-29 Update hash table versus Update 3-level indexed table

Insert/Delete hash table versus Insert/Delete 3-level indexed table

Figure 4-30 shows a Class 2 CPU reduction of 19% for inserting 1 row per commit and a 26% reduction for deleting 1 row per commit when comparing hash table against a 3-level indexed table.

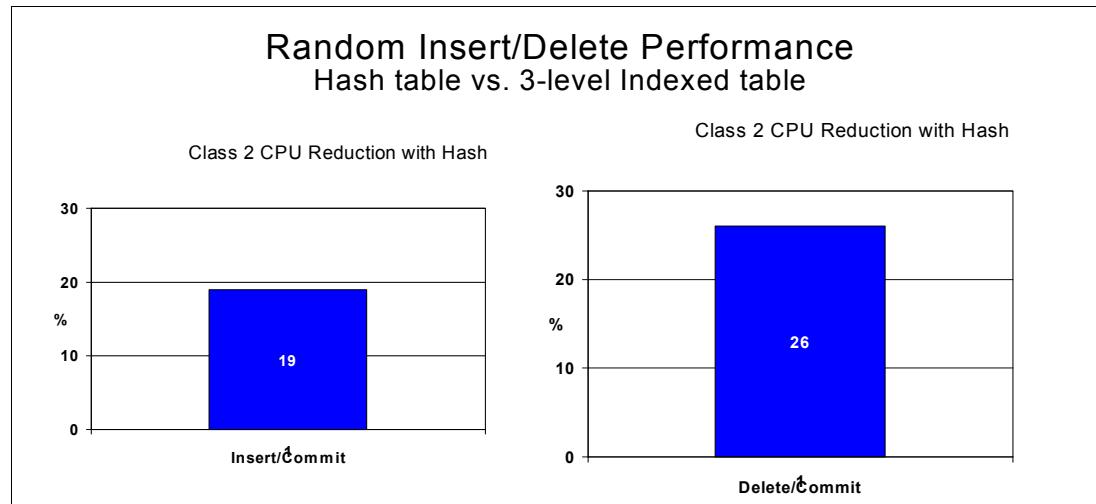


Figure 4-30 Insert/Delete Hash Table versus Insert/Delete 3-level indexed table

IRMM workload with hash tables

The intent of these measurements were to showcase the performance benefit of using hash access with the standard IRWW workload. IRWW is a warehouse operation, OLTP oriented. The workload consists of nine tables ranging from 800 rows to 240 million rows with both UTS PBG and PBR.

Five out of the original nine tables were converted to hash while keeping all indexes.

Seven different transactions issue 36 different SQL calls driven by JDBC Type 4 driver in dynamic SQL. 17 out of 36 SQL statements are hash access. No application change is needed.

The results are summarized in Figure 4-31.

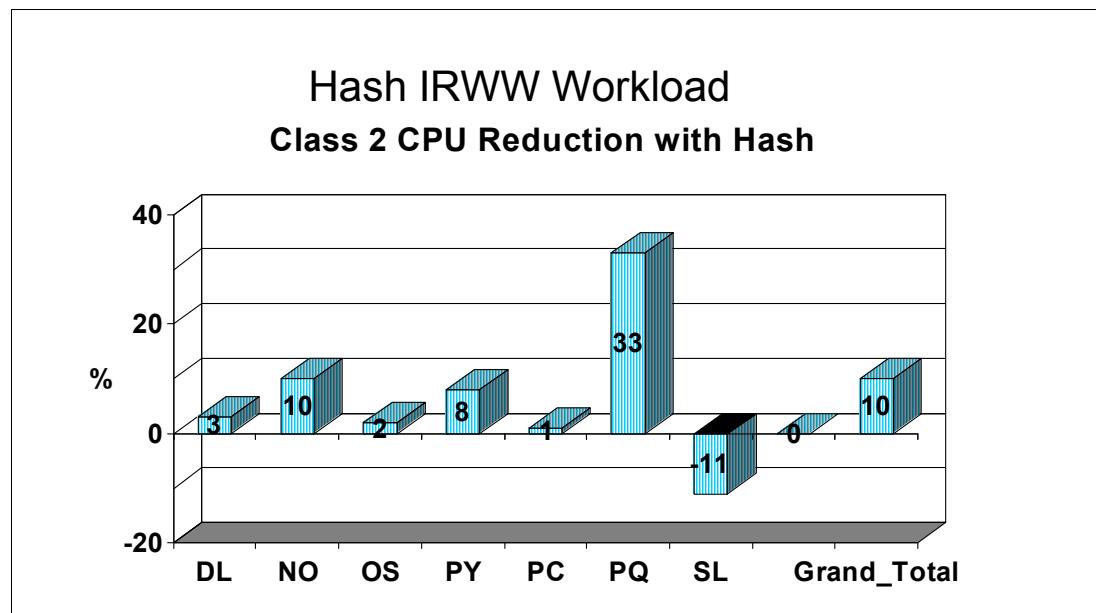


Figure 4-31 IRWW workload with hash tables

The hash IRWW workload measurements show that some application might not benefit at a great extent individually, but overall the workload performance is enhanced with hash, we observe:

- ▶ 10% reduction in class 2 CPU time compared to indexed IRWW
- ▶ Hash IRWW workload ITR is 7% higher than indexed IRWW
- ▶ An overall 10% class 2 CPU time reduction with hash
- ▶ Six transactions show a CPU reduction ranging from 1% to 33%

A larger percentage improvement can be realized if the applications are bound with RELEASE(DEALLOC)

The IRWW measurements demonstrate a critical aspect of hash access. When one table was converted to hash, one of the queries got slower, and yet the overall CPU performance for IRWW was better. The tradeoff is that certain queries will run slower in order to achieve an overall reduction in CPU utilization.

4.4.5 DDL and utilities

In this section we show the performance of hash table definition and how the DB2 utilities support hash.

Create table statement

The entire hash table has to be preformatted at create table time, which results in a long elapsed time. Table 4-2 shows the performance of creating a hash table with 10 partitions, each partition's hash space size is 2,250 MB on DS8300 and z10.

Table 4-2 Performance of CREATE a hash table.

	Indexed	Hash
Hash space		2,250 MB x 10
Elapsed time	7 sec	6 min. 09 sec
CPU time	0.09 sec	0.17 sec

Set DSNZPARM IMPDSDEF=YES (default). It defines the overflow index at create hash table time and avoids such penalty on the first insert to the overflow index (or REORG when converting to hash access.)

REORG

The REORG utility is enhanced to support hash access tables. The AUTOESTSPACE parameter directs the REORG utility to calculate the size of the hash space either by using the RTS values (AUTOESTSPACE YES, the default value) or by using the user-specified HASH SPACE values (AUTOESTSPACE NO) stored in SYSTABLESPACE and SYSTABLEPART. DB2 calculates the size of the hash space (when you specify AUTOESTSPACE YES) by estimating that about 5%-10% of the rows need to go into overflow. Note that automatic space calculation does not change the catalog values.

Although the REORG utility probably cannot remove all overflow entries, run it regularly to clean up the majority of overflow entries and reset hash chains, reducing the need to look into overflow. However, if your hash area is sized appropriately, then you might be able to run REORG less frequently.

Attention: DB2 10 provides an enhancement to DSNACCOX to suggest when to REORG on Hash Access. APAR PM25652 (currently open) provides this functionality. The specifics of the implementation might change between the time this book is published and when the APAR closes. Monitor the APAR for content and availability.

LOAD

The performance of LOAD utility with hash table is not equivalent to loading the data sequentially using the input in clustering order for traditional tables. With hash, the rows are loaded according to the randomized hash value and the space needs formatting.

Consider the following sequence for better performance when loading data into a hash table:

1. CREATE the table as non-hash.
2. LOAD the data.
3. ALTER the table to hash access.
4. REORG the entire table space.

Table 4-3 shows the time to load a STOCK table with 80 million rows spread over 10 partitions and 1 index.

Table 4-3 Loading a large hash table

Action	Time
CREATE without HASH	7 sec
LOAD	9 min. 57 sec
ALTER TABLE STOCK ADD ORGANIZE BY HASH	3 min. 40 sec
REORG with AUTOESTSPACE(YES)	24 min. 52 sec
Total	38 min. 36 sec

You need to execute REORG SHRLEVEL REFERENCE or CHANGE, NONE is not allowed in this process.

CHECK DATA

The DB2 check data utility ensures, in addition to normal check data functions, that the hash chains and the hash overflow indexes are correct.

CHECK INDEX

When you run the DB2 check index utility against a hash overflow index, DB2 validates consistency between the hash overflow index and the hash overflow area.

RECOVER

DB2 supports the RECOVER utility for a hash table space, with the exception that If you drop the hash organization (by ALTER), you cannot recover the table space to a point in time before the alter.

REBUILD INDEX for hash overflow index

When you run the DB2 REBUILD INDEX utility against a hash overflow index, DB2 scans only the hash overflow area for rows.

4.4.6 Monitoring the performance of hash access tables

The performance of hash tables is sensitive to the size of the hash space and to the number of rows that flow into overflow. If the fixed hash space is too small, then performance might suffer. Only 1 getpage is required to access the page pointed to by the hash routine. If there is no room on the page, then one or more getpages are required to access the overflow Index and another getpage is required to access the data in overflow. So, ongoing monitoring of space usage is important.

During insert, update, and delete operations, DB2 real time statistics (RTS) maintains a number of statistics in the SYSTABLESPACESTATS and SYSINDEXSPACESTATS catalog tables, as shown below. These statistics are relevant to monitoring the performance of hash access tables. These values are also used by the DB2 access path selection process to determine if using hash access is suitable.

- ▶ SYSIBM.SYSTABLESPACESTATS.TOTALROWS contains the actual number of rows in the table.
- ▶ SYSIBM.SYSTABLESPACESTATS.DATASIZE contains the total number of bytes used by the rows.
- ▶ SYSIBM.SYSINDEXSPACESTATS.TOTALENTRIES contains the total number of overflow rows.

The TOTALROWS and DATASIZE values apply for the whole table space, So, the HASH SPACE from the DDL when the hash table was created must be close to DATASIZE. Ideally, TOTALENTRIES must be less than 15% of TOTALROWS. If TOTALENTRIES is high, then you need to either ALTER the HASH SPACE or REORG the table space or let DB2 automatically recalculate the hash space upon next REORG.

- ▶ SYSIBM.STTABLESPACESTATS.REORGHASHACCESS records the number of times data is accessed using hash access for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints since the last CREATE, LOAD REPLACE, or REORG.
- ▶ SYSIBM.SYSINDEXSPACESTATS.REORGINDEXACCESS records the number of times DB2 has used the hash overflow index for SELECT, FETCH, searched UPDATE, searched DELETE, or used to enforce referential integrity constraints since the last CREATE, LOAD REPLACE, or REORG.

There is a possible savings in index maintenance by replacing an existing unique cluster index with an equivalent non-cluster index, allowing for faster insert and delete processing with the hash index. However, dropping of such an index must be done carefully by monitoring for some time real time statistics column SYSINDEXSPACESTATS.LASTUSED.

We showed significant performance improvement of random select, update, insert and delete with hash while we can keep the old indexes for DB2 to continue using index access. If you have fully qualified keys with equal predicates, then the hash can be used. Otherwise, for range predicates as an example, the indexes can be chosen.



Sample workloads

DB2 10 for z/OS continues to bring changes that improve performance keeping up with the explosive demands of transaction processing. DB2 delivers the ability to reduce CPU utilization and to remove virtual storage constraints, and adds functions that help insert intensive workloads.

In this chapter, we discuss the following topics:

- ▶ OLTP workloads
- ▶ Virtual and real storage
- ▶ INSERT performance improvements

5.1 OLTP workloads

In this section we summarize the performance changes observed during the execution of the following sample and customer workloads:

- ▶ DB2 10 and SAP on IBM System z
- ▶ IRWW workload

5.1.1 DB2 10 and SAP on IBM System z

For details on the performance and scalability provided by DB2 10 for z/OS in SAP environments, see the IBM white paper “*DB2 10 for z/OS with SAP on IBM System z Performance Report*”, available at the following website:

<http://www.ibm.com/support/techdocs/atstr.nsf/WebIndex/WP101845>

The IBM Poughkeepsie SAP Performance Evaluation on System z team ran DB2 10 with two different SAP workloads, SAP Sales and Distribution (SD) and SAP Banking Day Posting, with various configurations on a z10 and on a z196. This report is the result of over a year and a half evaluating the performance of DB2 10 drivers, prototypes, and tuning choices. The measurement tests were stress tests, not certified benchmarks.

The tests documented in this paper measure the performance of DB2 10 in terms of Internal Throughput Rate (ITR). For details, see the IBM web page, *Large Systems Performance Reference for IBM System z*, available at the following website:

<https://www.ibm.com/servers/resourcelink/lib03060.nsf/pages/lspindex?OpenDocument>

Using the SAP Sales and Distribution (SD) workload, this paper compares DB2 10 with DB2 9 on three different configurations of a z10. On a z10 2w, it measures a 3% improvement in ITR compared to DB2 9. On a z10 4w, there was a 12% improvement in ITR. And, on a z10 12w, there was a 19% improvement in ITR compared to DB2 9. As the DB2 system got larger, the performance improvement of DB2 10 grew. The details of these runs, using the SAP SD workload with DB2 9 and DB2 10 on a z10 2w, 4w, and 12w, are summarized in Table 5-1.

Table 5-1 SAP Sales and Distribution (SD) workload measurements

	DB2 9			DB2 10		
Server	z10	z10	z10	z10	z10	z10
Number of CPs	2	4	12	2	4	12
Number of users	3,735	6,480	14,400	3,735	6,480	14,400
Number of DB2 threads	144	279	409	144	279	409
MAXKEEPD (K)	8	8	8	8	8	8
% of CPU on z/OS	73.16	76.86	72.14	70.96	70.75	61.22
ETR (DS/sec)	365.26	615.35	1,421.19	365.27	636.05	1,422.19
ITR (DS/sec)	499.26	800.61	1,953.79	514.75	899.01	2,323.08
DBM1 below 2 GB bar (MB)	560	712	997	40	54	63

Figure 5-1 shows the ITR results from running the SAP SD workload with DB2 9 and DB2 10 at three N-way points (2w, 4w, and 12w) on a z10. On a z10 2w, the ITR with DB2 10 is 515 DS/sec compared to 499 DS/sec with DB2 9. On a z10 4w, the ITR with DB2 10 is 899 DS/sec compared to 801 DS/sec with DB2 9. The largest performance improvement is seen on the z10 12w, where the ITR with DB2 10 is 2,323 DS/sec compared to 1,954 DS/sec with DB2 9. This chart also shows the ITR delta between DB2 9 and DB2 10 at the three N-way points.

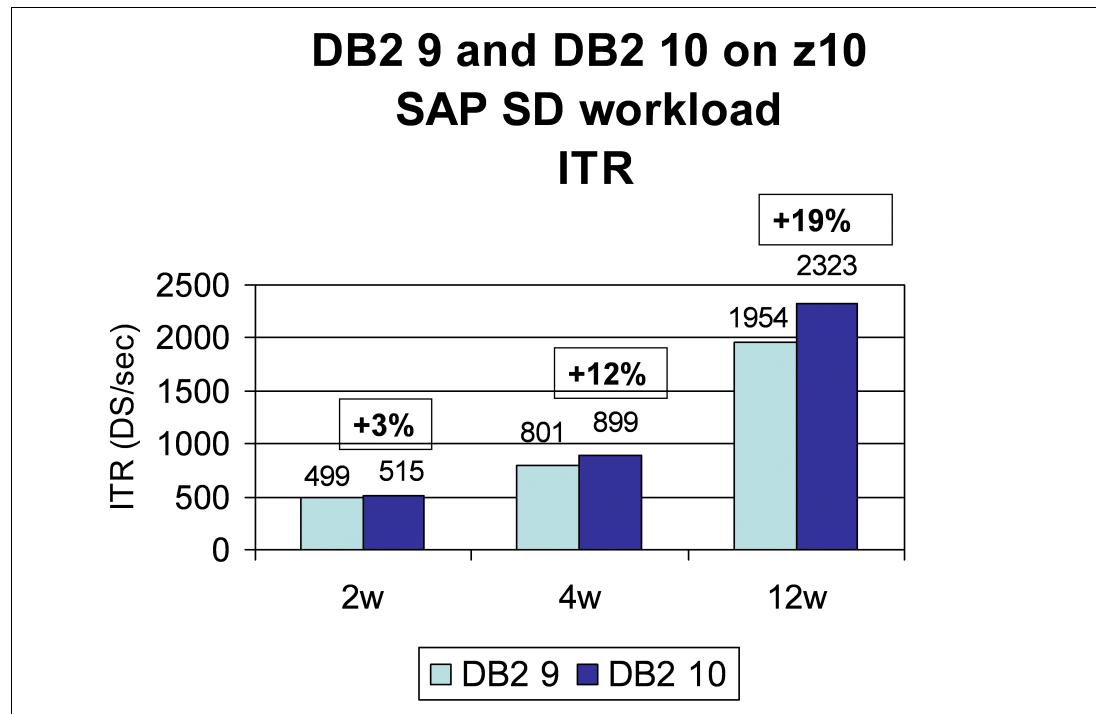


Figure 5-1 DB2 9 and DB2 10 ITR using SAP SD

Figure 5-2 shows the N-way scaling of DB2 9 and DB2 10 on z10 using the SAP SD workload. Looking at the ITR ratios of the SAP SD workload running on a z10 2w, 4w, and 12w, DB2 10 shows a better N-way performance scaling than DB2 9.

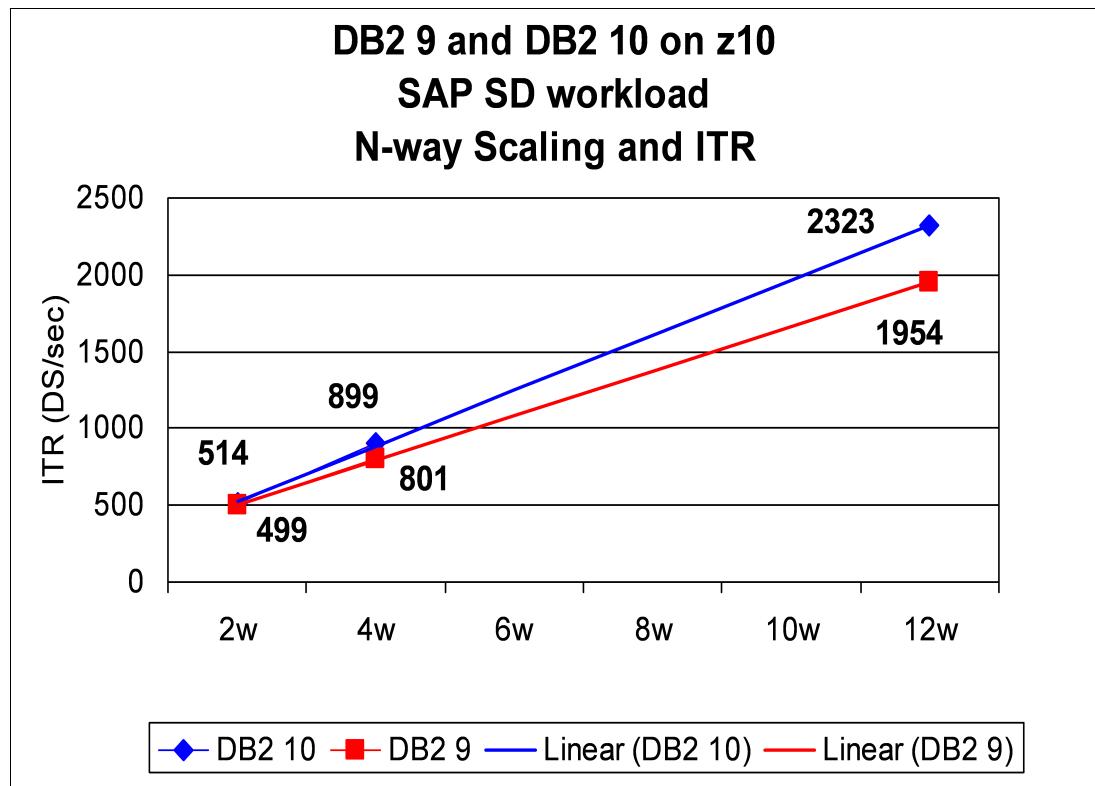


Figure 5-2 DB2 9 and DB2 10 N-way Scaling with SAP SD

This paper concludes by saying that measurement results show that DB2 10 for z/OS delivers significant performance and scalability improvements compared to DB2 9 on both the z10 and z196 systems. These results are from two different SAP workloads, the SAP Sales and Distribution (SD) workload and the SAP Banking Day Posting workload.

The DB2 10 for z/OS performance and scalability improvements seen in an SAP environment are summarized as follows:

- ▶ Basic performance improvements (such as latch class) result in reduced CPU costs related to size of DB2 system; this paper reports up to 19% more throughput.
- ▶ There are additional z196 improvements, up to 76% total.
- ▶ Significant Virtual Storage Constraint Relief is provided, up to 90% reduction:
 - Many more threads, measurements showed at least ~6x, per DB2 member, can result in fewer Parallel Sysplex members.
 - Maximum exploitation of MAXKEEPD for improved benefit from dynamic statement cache, that can lead to the following improvements:
 - Improved response time, measurements showed up to 74% reduction
 - Reduced CPU costs, measurements showed up to 46% more throughput
- ▶ More zIIP exploitation:
From 1 up to 5% zIIP redirect for asynchronous access in addition to DRDA

5.1.2 IRWW workload

IRWW is an OLTP workload that consists of seven transactions. Each transaction consists of one to many SQL statements, each performing a distinct business function in a predefined mix.

The seven transaction types, a brief description of each, and the percentage of the transaction mix are as follows:

- ▶ Neworder: Performs various SELECTS, FETCHES, UPDATES, and INSERTS in support of the receipt of new customer orders and runs as 22% of the total transaction mix.
- ▶ Order Status: Performs various SELECTS and FETCHES in support of providing the status of an order and runs as 24% of the total transaction mix.
- ▶ Payment: Performs SELECTS, FETCHES, UPDATES, and INSERTS in support of received customer payments and runs at 22% of the total transaction mix.
- ▶ Price Change: Performs an UPDATE in support of changing the price of an item and runs as 1% of the total transaction mix.
- ▶ Price Quote: Performs various SELECTS in support of providing the price of a set of items and runs as 25% of the total transaction mix.
- ▶ Stock Level: Performs a JOIN and various SELECTS in support of providing the current stock level of an item and runs at 4% of the mix.
- ▶ Delivery: Performs various SELECTS, UPDATES, and DELETES in support of the delivery of a group of orders and runs as 2% of the total transaction mix.

The IRWW workload is the base for a large series of tests executed with the goal of exhibiting the CPU and elapsed time characteristics of DB2 10.

This section document the following tests:

- ▶ IRWW OLTP performance
- ▶ IRWW distributed workloads
- ▶ IRWW OLTP workload with UTS

IRWW OLTP performance

This section documents the observed results of an IRWW IMS DB2 OLTP workload simulating a warehouse operation. IMS (Information Management System) is the transaction system. The following list briefly describes the testing environment:

- ▶ Data sharing and non-data sharing environment
- ▶ z10 processor
- ▶ z/OS 1 MB pages
- ▶ Buffer pools are pagefixed

The DB2 10 subsystem on which these tests were executed was running with APAR PM31614 (PTF UK66374), which provides package allocation improvement.

The table presented in Figure 5-3 shows the performance comparing DB2 9 to DB2 10.

The ITR (Internal Throughput Rate) is measured in number of commits per second. DB2 10 shows an increase in ITR of 3.3%.

The CPU time is reported per commit. Class 2 CPU time is reduced by 3% in DB2 10. Considering the DB2 address spaces total CPU cost per commit, DB2 10 improves the resource utilization by showing a reduction of 6.1%.

Measurement	DB2 9	DB2 10	% Delta
CPU %	48.93	47.57	
ITR (commit/sec)	1543.36	1594.14	+ 3.3 %
CPU TIME (msec/commit)			
CLASS 2 Time	0.691	0.670	- 3.0 %
MSTR	0.041	0.036	
DBM1	0.098	0.073	
IRLM	0	0	
TOTAL	0.830	0.779	- 6.1 %

Figure 5-3 IRWW OLTP non-data sharing measurements

The same sample workload was also executed in a data sharing environment. The table presented in Figure 5-4 shows the results when the application was executed with RELEASE(COMMIT). This is the same behavior as available in DB2 9. Under the same considerations listed for the non data sharing tests, DB2 10 provides about 3% more throughput with a Class 2 CPU time reduction of approximately 2%. When considering total CPU utilization, including all address spaces, the CPU reduction is about 5%.

	DB2 9	DB2 9	DB2 10	DB2 10	% Delta
Member	1	2	1	2	
CPU %	41.52	40.14	40.76	38.97	
ITR (commit/sec)	1056.92	1096.03	1093.39	1122.23	+ 2.9%
CPU TIME (msec/commit)					
CLASS 2 Time	1.025	1.013	1.005	0.990	- 2.1 %
MSTR	0.064	0.064	0.058	0.057	
DBM1	0.159	0.146	0.123	0.112	
IRLM	0.006	0.006	0.006	0.006	
TOTAL	1.254	1.229	1.192	1.165	- 5.1 %

Figure 5-4 IRWW OLTP data sharing measurements, RELEASE(COMMIT)

RELEASE(COMMIT) is the only supported option for distributed applications in DB2 9 and is often the preferred option for local applications (such as IMS) when virtual storage constrained. DB2 10 allows the use of RELEASE(DEALLOCATE) in local and in distributed environments. This feature can provide further CPU improvements. For distributed applications, see Chapter 8, “Distributed environment” on page 239 for more details.

Figure 5-5 shows the results of the same test with DB2 10 exploiting RELEASE(DEALLOCATE). As expected, there is an increase in throughput and a further CPU reduction. When considering all address spaces, the CPU reduction was reported as 12.6%.

	DB2 9	DB2 9	DB2 10	DB2 10	% Delta
Member	1	2	1	2	
CPU %	41.52	40.14	38.47	37.09	
ITR (commit/sec)	1056.92	1096.03	1154.08	1182.71	+ 8.5%
CPU TIME (msec/commit)					
CLASS 2 Time	1.025	1.013	0.870	0.879	- 14.1 %
MSTR	0.064	0.064	0.064	0.063	
DBM1	0.159	0.146	0.143	0.132	
IRLM	0.006	0.006	0.009	0.009	
TOTAL	1.254	1.229	1.086	1.083	- 12.6 %

Figure 5-5 IRWW OLTP data sharing measurements, RELEASE(DEALLOCATE)

Results will vary because they are workload dependent. An increase in throughput observed at the same time as a CPU reduction can be an indication of CPU being the limiting factor. In circumstances where CPU is not the limiting factor, for instance, when the application is I/O bound, the CPU reduction might not translate into an increased throughput.

Figure 5-6 shows the virtual storage utilization for the non-data sharing, data sharing, and DB2 9, DB2 10 scenarios as measured during the execution of the test described in this section. DB2 10 shows a dramatic relief on storage utilization below the bar.

	DB2 9	DB2 10	% Delta
	Non DS	Non DS	
	DS	DS	
Total DBM1 Storage below 2 GB (MB)	83.78	19.89	- 76 %
	92.23	30.00	- 67 %
Total Get Main Storage (MB)	22.33	1.82	- 92 %
	25.01	4.11	- 84 %
EDM Pool (MB)	19.53	0	- 100 %
	19.53	0	- 100 %
Total Agent System Storage (MB)	25.92	4.15	- 84 %
	23.86	2.60	- 89 %
Total Agent Non-System Storage (MB)	10.11	1.27	- 87 %
	11.04	1.53	- 86 %
Total Get Main Stack Storage (MB)	20.99	11.21	- 47 %
	26.96	14.62	- 46 %

Figure 5-6 IRWW OLTP Non DS and DS, DBM1 and MVS storage below 2 GB bar

This section reports the performance and storage measurements for several IRWW OLTP scenarios including non data sharing and data sharing environments.

DB2 10 shows in each scenario an increase in throughput with a CPU reduction. The tests showed an important reduction on storage utilization below the 2 GB bar.

IRWW OLTP workload with UTS

We executed the IRWW IMS OLTP workload on non data sharing environment on a z10.

As the base case scenario (A) we used the original classic partitioned table spaces and segmented table spaces with DB2 9.

For the second scenario (B) we executed the same workload with DB2 10.

For the third scenario (C) we converted the classic partition table space to PBR using SEGSIZE 32. For the partitioned table spaces, the indexes were re-created with the same key values, but without the PARTITIONED keyword. We also converted the segmented table spaces to PBG using MAXPART 1. The indexes contain the same key values. We then ran REORG and RUNSTATS to remove the pending changes and supply statistics.

Table 5-2 shows the results comparing the three scenarios for this test. CPU times are in msec.

Table 5-2 Classic versus UTS table space measurements for non-data sharing

	A) DB2 9 SEG/CLASSIC	B) DB2 10 SEG/CLASSIC	%B/A DELTA	C) DB2 10 PBG/PBR	%C/A DELTA
CPU %	48.18	47.08		47.25	
ITR (commit/sec)	1594.02	1636.93	+2.7%	1625.40	+2.0%
CLASS 2 time	0.683	0.655	-4.1%	0.657	-3.8%
MSTR	0.039	0.038		0.038	
DBM1	0.083	0.069		0.069	
IRLM	0	0		0	
TOTAL CPU	0.805	0.762	-5.3%	0.764	-5.1%

Going from DB2 9 to DB2 10 with traditional table spaces shows a small improvement in ITR and a CPU reduction. The performance numbers in DB2 10 between traditional and UTS table spaces show equivalent performance and very similar improvements from DB2 9.

IRWW distributed workloads

Several tests of distributed access to DB2 for z/OS were conducted focusing on the differences between DB2 9 and DB2 10. We used the IRWW workload with 7 transactions, 3 of which were read only, in the following test environment:

- ▶ z10 z/OS LPAR: 3 CPs, 32 GB, z/OS 1.11
- ▶ z Linux LPAR: 2 CP
- ▶ DB2 Connect V9.7 Fix Pack 3A
- ▶ T4 JCC driver 3.59.52, JDK 1.6
- ▶ IBM HiperSockets™ communication between the two LPARs

During this test, we executed and reported the performance of the following connectivity options:

- ▶ SQCL: SQL ODBC / CLI (dynamic)
- ▶ SPCB: Stored procedures in COBOL (static)
- ▶ JCC T4 Driver (DDF):
 - JDBC: Dynamic SQL
 - SQLJ: Static SQL
 - SPSJ: Stored procedures in SQLJ with static SQL
 - SPNS: Stored procedures in native SQL static

DB2 9 versus DB2 10 total CPU

The reported total transaction CPU for SQCL, JDBC, SQLJ, and SPNS workloads is the sum of System Services Address Space, Database Services Address Space, IRLM, and DDF Address Space CPU time per commit as reported in OMEGAMON PE statistics report Address Space CPU section. The reported total transaction CPU for stored procedure workloads such as SPCB and SPCJ is the statistics total address space CPU per commit plus accounting class 1 STORED PRC CPU in the OMEGAMON PE report.

Because DB2 9 works only with RELEASE(COMMIT) for distributed applications, we used RELEASE(COMMIT) in DB2 10 as well for getting comparable results.

Figure 5-7 shows the observed results. This chart reports the total CPU time, in seconds, per transaction comparing DB2 9 to DB2 10.

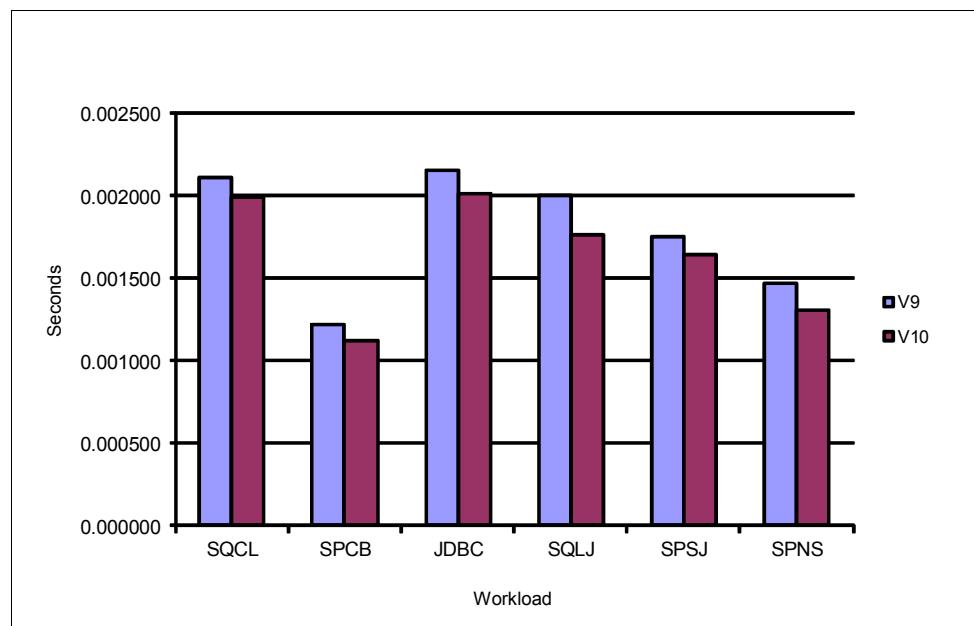


Figure 5-7 Total CPU time, DB2 9 versus DB2 10 compared for distributed IRWW

The results show that the improvement varies between a 5% to a 12% reduction in total CPU time.

DB2 9, DB2 10 RELEASE(COMMIT), and DB2 10 RELEASE(DEALLOCATE)

Further CPU reduction can be achieved with DB2 10 for distributed applications when changing the BIND to RELEASE(DEALLOCATE). Table 5-3 shows the same values of Figure 5-7 and adds the benefits of RELEASE(DEALLOCATE) in CPU time reported in microseconds (this option is only available in DB2 10).

Table 5-3 RELEASE(COMMIT) versus RELEASE(DEALLOCATE) for distributed applications

Total CPU transaction (microsec.)	DB2 9	DB2 10 PKREL(COMMIT)	Delta %	DB2 10 PKREL(BNDOPT)	Delta %
SQCL	2114	1997	-5.5	1918	-9.3
SPCB	1221	1124	-7.9	1056	-13.5
JDBC	2152	2017	-6.3	1855	-13.8
SQLJ	1899	1761	-11.9	1689	-16.6
SPSJ	1768	1642	-6.7	1550	-11.9
SPNS	1472	1304	-11.4	1180	-19.8

Using the same distributed workload, we measured the DB2 10 effects on CPU when using RELEASE(COMMIT) and RELEASE(DEALLOCATE). Distributed packages were bound using the RELEASE(DEALLOCATE) BIND option and we controlled the high performance DBATs behavior by means of the new -MODIFY DDF PKGREL command described in “The MODIFY DDF PKGREL BNDOPT command” on page 241.

DB2 10 RELEASE(COMMIT) versus RELEASE(DEALLOCATE): Class 1 and 2 CPU time

We now zoom on the Class 1 and Class 2 CPU time on the two groups of workloads previously shown only as total CPU time. The CPU time is in microseconds.

The values in the x axis are as follows:

► COMMIT

The test was executed using PKGREL = COMMIT. The rules of the RELEASE(COMMIT) bind option are applied to any package that is used for remote client processing.

► BNDOPT

The test was executed using PKGREL = BNDOPT. The rules of the RELEASE bind option that was specified when the package was bound are applied to any package that is used for remote client processing. BNDOPT is the default value of the MODIFY DDF PKGREL command.

For COBOL, SQLJ, and native stored procedures, the results are documented in Figure 5-8.

The test showed these improvements:

- Class 1 CPU time is reduced between 4.5% to 8.7%
- Class 2 CPU time is reduced between 5.0% to 8.9%

Native SQL stored procedures show the largest improvement.

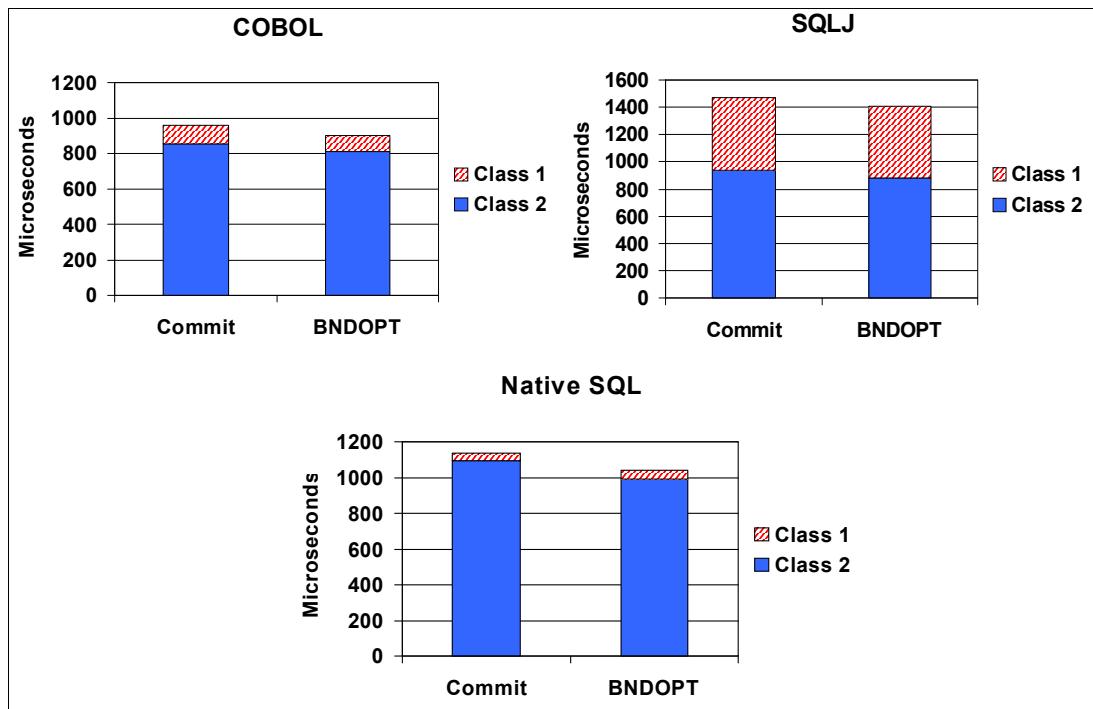


Figure 5-8 DB2 10 distributed stored procedures, CPU and PKGREL

For CLI, SQLJ, and JDBC T4 distributed workloads, equivalent series of tests were done. Figure 5-9 shows the resulting changes in Class 1 and Class 2 CPU Time.

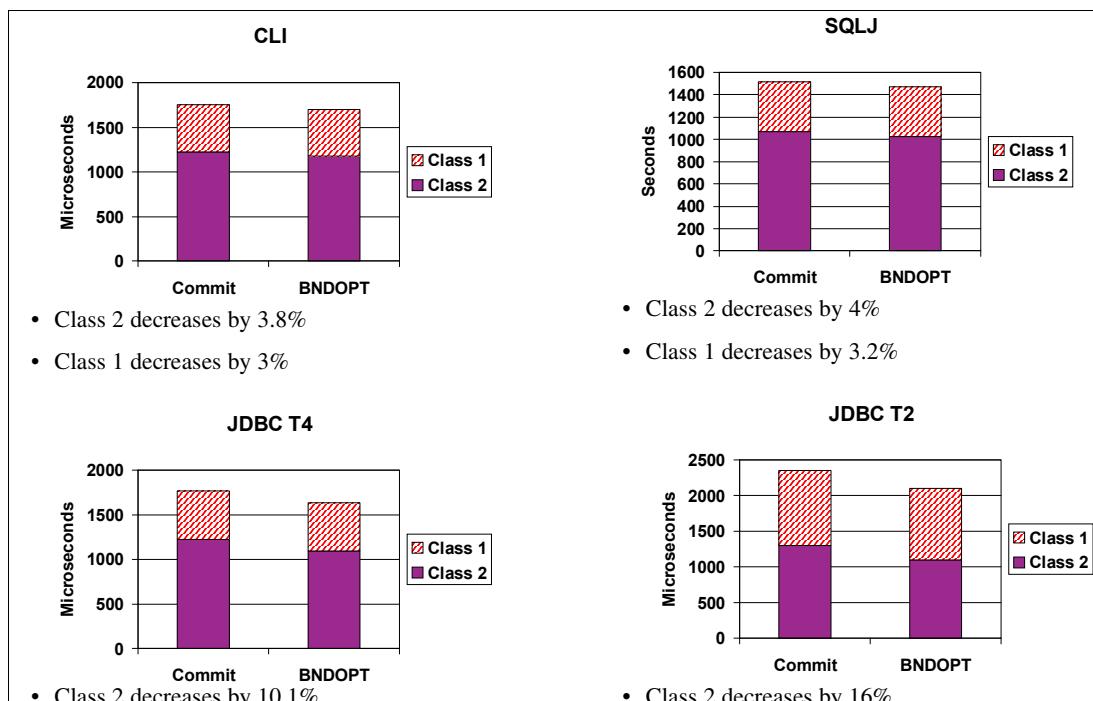


Figure 5-9 DB2 10 Distributed Workloads, CPU and PKGREL

These observations can be summarized as follows:

- ▶ CLI:
 - Class 1 CPU Time reduced by 3%
 - Class 2 CPU Time reduced by 3.8%
- ▶ SQLJ:
 - Class 1 CPU Time reduced by 3.2%
 - Class 2 CPU Time reduced by 4%
- ▶ JDBC T4:
 - Class 1 CPU Time reduced by 7.4%
 - Class 2 CPU Time reduced by 10.1%

With distributed workloads, SQL statements and stored procedures show reductions in CPU time when comparing DB2 9 and DB2 10 without requiring changes in the applications.

The implementation of RELEASE(DEALLOCATE) further improves the benefits. Results will vary and the observed performance is dependent on the running conditions and the workload characteristics.

See Chapter 8, “Distributed environment” on page 239 for details about DB2 10 and RELEASE(DEALLOCATE) for distributed workloads.

5.2 Virtual and real storage

Prior to DB2 V8, the primary constraint to vertical scalability was virtual storage below the 2 GB bar.

DB2 V8 introduced the initial 64-bit support for DBM1 and IRLM address spaces. DB2 moved many of its data areas, control blocks, and buffer pools above the bar. In some cases, DB2 V8 achieved better performance because moving the buffer pool control blocks above 2 GB, DB2 V8 can manage a much larger buffer pool than DB2 V7. However, DB2 V8 was not designed to manage more threads than DB2 V7.

DB2 9 for z/OS provided an additional 10-15% relief for agent thread usage in DBM1 and moved the distributed data facility (DDF) address space to run above the bar. It also introduced the use of Shared Memory. The shared memory object is created at DB2 startup, and all DB2 address spaces for the subsystem (DIST, DBM1, MSTR, and Utilities) are registered to be able to access the shared memory object.

DB2 10 for z/OS has moved up to 90% of the DBM1 address space required storage above the bar, as shown on Figure 5-10.

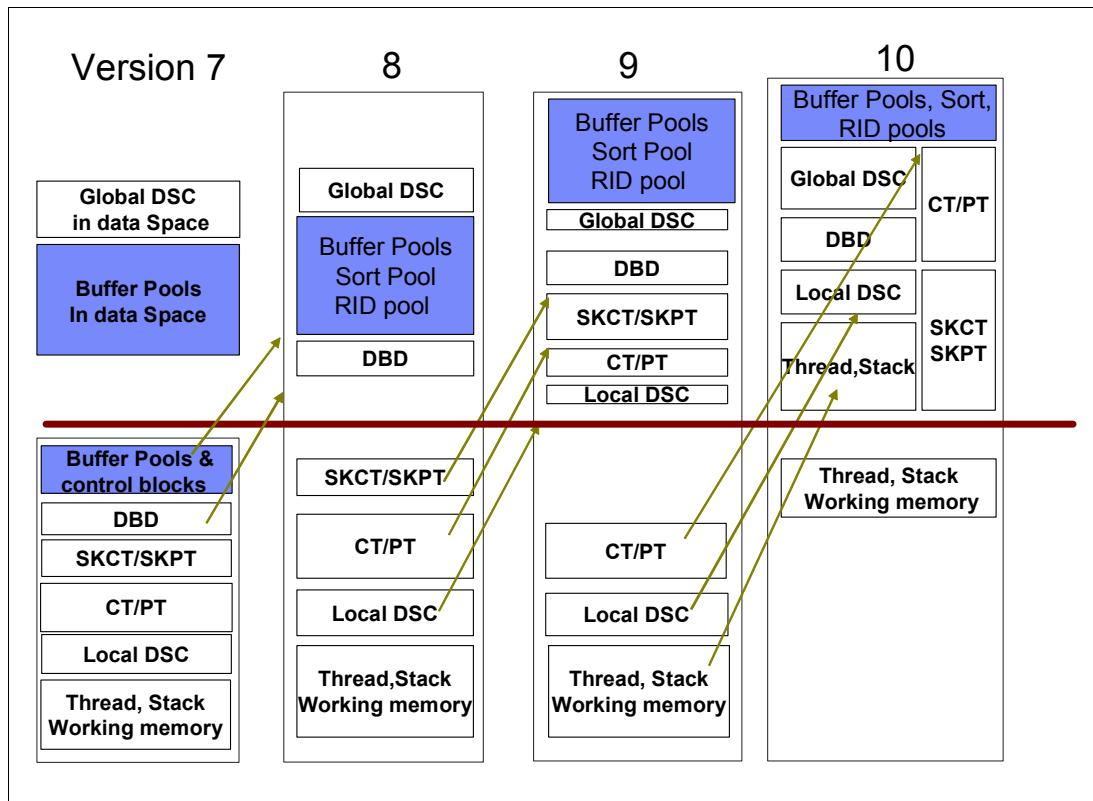


Figure 5-10 DBM1 address space memory relief across versions

The changes to thread related virtual storage in DB2 10 that increase scalability are as follows:

- ▶ Most control blocks are now allocated above-the-bar.
- ▶ Rebound applications have plan and package structures allocated above-the-bar.
- ▶ Column procedures (SPROC, IPROC, and so on) are allocated and shared below-the-bar for applications rebound on DB2 10.
- ▶ SPROC is no longer limited by the amount of 4 KB storage. SPROC is enabled for up to 750 columns instead of about 100 columns maximum, with a significant CPU reduction when fetching many columns.
- ▶ Large fixed areas for EDM thread pools are eliminated with plan and package structures allocated in thread storage pools.
- ▶ Stack storage is split between above-the-bar and below-the-bar portions.

For applications running on DB2 10 but bound on a prior release, the following considerations apply:

- ▶ They still have plan and package structures allocated below-the-bar.
- ▶ Some control blocks are “puffed” at execution below-the-bar for compatible DB2 10 interfaces in case of fall-back.
- ▶ Column procedures are regenerated for each thread’s usage and are not shared among common users of the same statement.

5.2.1 Common storage and real storage

For reference, in Figure 5-11 we have provided the layout of z/OS virtual storage.

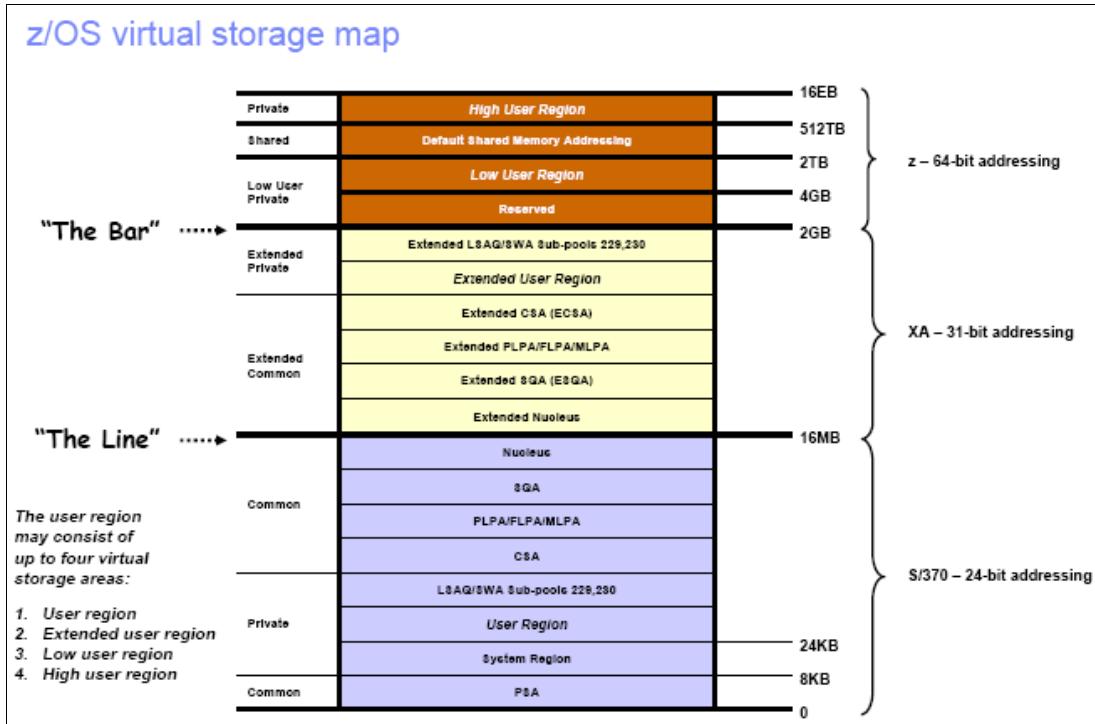


Figure 5-11 Layout of z/OS virtual storage

DB2 is allocating all buffer pools and most of its pools and control blocks above the 2 GB bar, with the theoretical possible expansion to 16 exabytes.

Practically, the use of 64-bit addressing has removed the limitations of 2 GB for each DB2 address space. This has removed the constraints to grow DB2 workloads vertically within an LPAR. However, DB2 has always advised not to exceed the availability of real storage when defining pools in order to avoid paging. It is also important, especially when looking at consolidating DB2 subsystems to check the use of z/OS common storage. Generally, from release to release, the use of real storage by DB2 has increased by 10 to 20%.

DB2 uses Private and Shared storage. Buffer pools are allocated in 64-bit Private, the majority of local thread and stack storage is now in 64-bit Shared. Many common control blocks and distributed threads have moved to HCSA and Private above the bar but some are still allocated in ECSA.

Because there is a renewed focus with DB2 10 towards carefully planning, provisioning, and monitoring real storage and common storage (ECSA and ESQA 31-bit) consumption, there are new statistics in IFCID 225 to report:

- ▶ DBM1 and DIST address space: virtual storage below and above bar, real, and auxiliary
- ▶ Common storage and 64-bit shared storage usage

Monitoring has been enhanced to collect accurate 64-bit shared and common when running multiple DB2 subsystems on the same LPAR. MVS APAR OA35885 implements new callable service to RSM to provide REAL and AUX for addressing range for shared objects and DB2 APARs PM24723 and PM37647 provide new IFCID 225 counters and other functions to include values on real storage statistics.

z/OS APAR OA33106 also reduced ESQA storage usage for SRBs using linkage stack and their status is being saved.

Examples of the new report layouts from a distributed workload with 1000 threads are shown here. The new text is reported as blue.

Example 5-1 shows the report on DBM1 and MVS storage used by the workload below the 2 GB bar with the new thread information.

Example 5-1 DBM1 storage below the 2 GB bar information

DBM1 AND MVS STORAGE BELOW 2 GB	QUANTITY	DBM1 AND MVS STORAGE BELOW 2 GB	CONTINUED	QUANTITY
TOTAL DBM1 STORAGE BELOW 2 GB (MB)	86.70	24 BIT LOW PRIVATE (MB)	0.21	
TOTAL GETMAINED STORAGE (MB)	4.40	24 BIT HIGH PRIVATE (MB)	1.23	
EDM POOL (MB)	0.00	24 BIT PRIVATE CURRENT HIGH ADDRESS (MB)	000000000003C000	
TOTAL VARIABLE STORAGE (MB)	38.83	31 BIT EXTENDED LOW PRIVATE (MB)	67.71	
TOTAL AGENT LOCAL STORAGE (MB)	32.84	31 BIT EXTENDED HIGH PRIVATE (MB)	113.42	
TOTAL AGENT SYSTEM STORAGE (MB)	5.32	31 BIT PRIVATE CURRENT HIGH ADDRESS (MB)	000000001DA2D000	
NUMBER OF PREFETCH ENGINES	148	EXTENDED REGION SIZE (MAX) (MB)	1665.00	
NUMBER OF DEFERRED WRITE ENGINES	300	EXTENDED CSA SIZE (MB)	256.82	
NUMBER OF CASTOUT ENGINES	0	AVERAGE THREAD FOOTPRINT (MB)	0.07	
NUMBER OF GBP WRITE ENGINES	0	MAX NUMBER OF POSSIBLE THREADS	16548	
NUMBER OF P-LOCK/NOTIFY EXIT ENGINES	0			
TOTAL AGENT NON-SYSTEM STORAGE (MB)	27.52			
TOTAL NUMBER OF ACTIVE USER THREADS	1000	AVERAGE THREAD FOOTPRINT (TYPE II) (MB)	0.06	
NUMBER OF ALLIED THREADS	0	MAX NUMBER OF POSSIBLE THREADS (TYPE II)	19306	
NUMBER OF ACTIVE DBATS	1000			
NUMBER OF POOLED DBATS	0			
NUMBER OF PARALLEL CHILD THREADS	0			
RID POOL (MB)	N/A			
PIPE MANAGER SUB POOL (MB)	N/A			
LOCAL DYNAMIC STMT CACHE CNTL BLKS (MB)	N/A			
THREAD COPIES OF CACHED SQL STMTS (MB)	5.27			
IN USE STORAGE (MB)	0.20			
STATEMENTS COUNT	N/A			
HWM FOR ALLOCATED STATEMENTS (MB)	0.21			
STATEMENT COUNT AT HWM	N/A			
DATE AT HWM	N/A			
TIME AT HWM	N/A			
THREAD COPIES OF STATIC SQL (MB)	0.00			
IN USE STORAGE (MB)	0.00			
THREAD PLAN AND PACKAGE STORAGE (MB)	0.00			
BUFFER MANAGER STORAGE CNTL BLKS (MB)	0.00			
TOTAL FIXED STORAGE (MB)	0.46			
TOTAL GETMAINED STACK STORAGE (MB)	43.00			
TOTAL STACK STORAGE IN USE (MB)	41.14			
SYSTEM AGENT STACK STORAGE IN USE (MB)	9.83			
STORAGE CUSHION (MB)	356.96			

Example 5-2 shows the DBM1 above the 2 GB bar layout with the new shared storage information.

Example 5-2 DBM1 above the 2 GB bar storage layout

DBM1 STORAGE ABOVE 2 GB	QUANTITY
GETMAINED STORAGE (MB)	1507.34
FIXED STORAGE (MB)	11.16
VARIABLE STORAGE (MB)	242.93
COMPRESSION DICTIONARY (MB)	0.00
IN USE EDM DBD POOL (MB)	0.42
IN USE EDM STATEMENT POOL (MB)	1.54
IN USE EDM RDS POOL (MB)	N/A
IN USE EDM SKELETON POOL (MB)	0.11
STAR JOIN MEMORY POOL (MB)	N/A
STORAGE MANAGER CONTROL BLOCKS (MB)	2.34
VIRTUAL BUFFER POOLS (MB)	8843.98
VIRTUAL POOL CONTROL BLOCKS (MB)	315.17
CASTOUT BUFFERS (MB)	0.00
SHARED GETMAINED STORAGE (MB)	4.09
SHARED FIXED STORAGE (MB)	49.76
RID POOL (MB)	5.00
SHARED VARIABLE STORAGE (MB)	2412.13
TOTAL AGENT LOCAL STORAGE (MB)	2088.91
TOTAL AGENT SYSTEM STORAGE (MB)	32.69
TOTAL AGENT NON-SYSTEM STORAGE (MB)	2056.22
DYNAMIC STMT CACHE CNTL BLKS (MB)	5.37
THREAD COPIES OF CACHED SQL STMTS (MB)	N/A
IN USE STORAGE (MB)	24.97
STATEMENTS COUNT	2380
HWM FOR ALLOCATED STATEMENTS (MB)	29.19
STATEMENT COUNT AT HWM	2776
DATE AT HWM	05/10/11
TIME AT HWM	04:08:24.62
THREAD PLAN AND PACKAGE STORAGE (MB)	90.91
SHARED STORAGE MANAGER CNTL BLKS (MB)	53.52
SHARED SYSTEM AGENT STACK STORAGE (MB)	512.00
STACK STORAGE IN USE (MB)	130.00
SHARED NON-SYSTEM AGENT STACK STORAGE (MB)	1536.00
STACK STORAGE IN USE (MB)	500.00

Example 5-3 shows the new distributed address space storage layout below and above the 2 GB bar information.

Example 5-3 Distributed address space storage below and above the 2 GB bar

DIST AND MVS STORAGE BELOW 2 GB		QUANTITY	DIST STORAGE ABOVE 2 GB		QUANTITY
TOTAL DIST STORAGE BELOW 2 GB	(MB)	134.47	FIXED STORAGE	(MB)	1.00
TOTAL GETMAINED STORAGE	(MB)	0.03	GETMAINED STORAGE	(MB)	0.00
TOTAL VARIABLE STORAGE	(MB)	11.77	VARIABLE STORAGE	(MB)	42.02
NUMBER OF ACTIVE CONNECTIONS		1000	STORAGE MANAGER CONTROL BLOCKS	(MB)	8.59
NUMBER OF INACTIVE CONNECTIONS		0			
TOTAL FIXED STORAGE	(MB)	0.88			
TOTAL GETMAINED STACK STORAGE	(MB)	121.80			
TOTAL STACK STORAGE IN USE	(MB)	121.80			
SYSTEM AGENT STACK STORAGE IN USE	(MB)	16.23			
STORAGE CUSHION	(MB)	357.83			
24 BIT LOW PRIVATE	(MB)	0.23			
24 BIT HIGH PRIVATE	(MB)	0.21			
24 BIT PRIVATE CURRENT HIGH ADDRESS		0000000000042000			
31 BIT EXTENDED LOW PRIVATE	(MB)	5.14			
31 BIT EXTENDED HIGH PRIVATE	(MB)	148.67			
31 BIT PRIVATE CURRENT HIGH ADDRESS		0000000018425000			
EXTENDED REGION SIZE (MAX)	(MB)	1665.00			

Example 5-4 shows the report on real and auxiliary storage information for DBM1 and the new section on DIST address space.

Example 5-4 Real and auxiliary storage information for DBM1 and DIST address spaces

REAL AND AUXILIARY STORAGE FOR DBM1		QUANTITY	REAL AND AUXILIARY STORAGE FOR DIST		QUANTITY
REAL STORAGE IN USE	(MB)	9759.86	REAL STORAGE IN USE	(MB)	92.25
31 BIT IN USE	(MB)	166.55	31 BIT IN USE	(MB)	63.58
64 BIT IN USE	(MB)	9593.31	64 BIT IN USE	(MB)	28.67
64 BIT THREAD AND SYSTEM ONLY	(MB)	395.82	64 BIT THREAD AND SYSTEM ONLY	(MB)	28.65
HWM 64 BIT REAL STORAGE IN USE	(MB)	9593.31	HWM 64 BIT REAL STORAGE IN USE	(MB)	28.67
AVERAGE THREAD FOOTPRINT	(MB)	0.56	AVERAGE DBAT FOOTPRINT	(MB)	0.09
AUXILIARY STORAGE IN USE	(MB)	0.00	AUXILIARY STORAGE IN USE	(MB)	0.00
31 BIT IN USE	(MB)	0.00	31 BIT IN USE	(MB)	0.00
64 BIT IN USE	(MB)	0.00	64 BIT IN USE	(MB)	0.00
64 BIT THREAD AND SYSTEM ONLY	(MB)	0.00	64 BIT THREAD AND SYSTEM ONLY	(MB)	0.00
HWM 64 BIT AUX STORAGE IN USE	(MB)	0.00	HWM 64 BIT AUX STORAGE IN USE	(MB)	0.00

Example 5-5 shows the new common and subsystem shared information report.

Example 5-5 Common and subsystem shared storage report

COMMON STORAGE BELOW AND ABOVE 2 GB		QUANTITY	SUBSYSTEM SHARED STORAGE ABOVE 2 GB		QUANTITY
EXTENDED CSA SIZE	(MB)	256.82	REAL STORAGE IN USE	(MB)	2302.40
FIXED POOL BELOW	(MB)	6.93	SHARED THREAD AND SYSTEM	(MB)	2060.78
VARIABLE POOL BELOW	(MB)	1.04	SHARED STACK STORAGE	(MB)	241.62
GETMAINED BELOW	(MB)	0.07	AVERAGE THREAD FOOTPRINT	(MB)	2.30
FIXED POOL ABOVE	(MB)	10.27	AUXILIARY STORAGE IN USE	(MB)	0.00
VARIABLE POOL ABOVE	(MB)	0.00	SHARED THREAD AND SYSTEM	(MB)	0.00
GETMAINED ABOVE	(MB)	0.00	SHARED STACK STORAGE	(MB)	0.00
STORAGE MANAGER CONTROL BLOCKS ABOVE	(MB)	1.34			
REAL STORAGE IN USE	(MB)	11.48			
AVERAGE THREAD FOOTPRINT	(MB)	0.01			
AUXILIARY STORAGE IN USE	(MB)	0.00			
MVS LPAR SHARED STORAGE ABOVE 2 GB		QUANTITY			QUANTITY
SHARED MEMORY OBJECTS		2			
64 BIT SHARED STORAGE	(MB)	163840.00			
HWM FOR 64 BIT SHARED STORAGE	(MB)	491520.00			
64 BIT SHARED STORAGE BACKED IN REAL	(MB)	2302.41			
AUX STORAGE USED FOR 64 BIT SHARED	(MB)	0.00			
64 BIT SHARED STORAGE PAGED IN FROM AUX	(MB)	0.00			
64 BIT SHARED STORAGE PAGED OUT TO AUX	(MB)	0.00			

As part of the statistics report, DB2 also provides the following information:

SHORT-ON-STORAGE, METRICS	QUANTITY	/SECOND	/THREAD	/COMMIT
FULL SYSTEM CONTRACTIONS	0	0.00	N/C	0.00
CRITICAL SHORTAGES	0	0.00	N/C	0.00
ABENDS DUE TO SHORTAGES	0	0.00	N/C	0.00

For details, see *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Version 5.1.0 Report Reference*, SH12-6921.

5.2.2 Subsystems consolidation

As a result of these memory relief changes, assuming that there are no latch constraints (see 2.2, “Latching contention relief” on page 22) or other configuration issues (such as I/O or real storage), DB2 10 is able to run much more concurrent work in a single subsystem and to run from five to ten times more concurrent threads as shown in Figure 5-12. For instance, if your configuration and application mix can support 500 concurrent threads on a single subsystem with DB2 9, you can support as many as 5000 concurrent threads on a single subsystem with DB2 10.

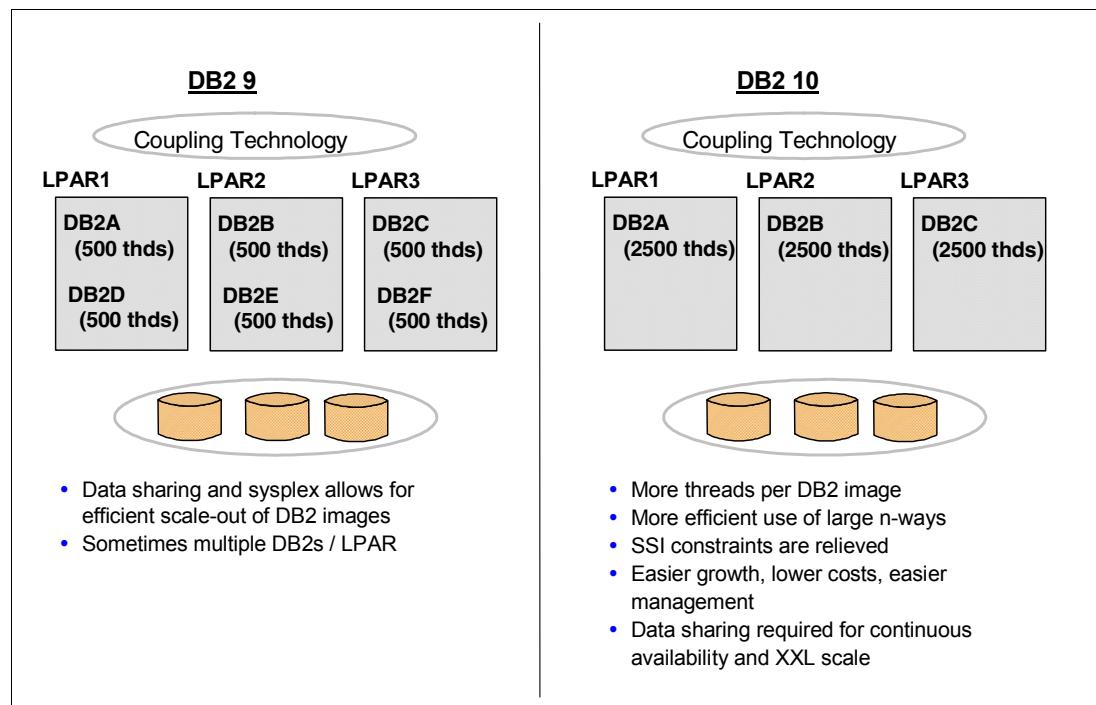


Figure 5-12 DB2 running large number of threads

With the change in virtual storage in DB2 10, more work can run in one DB2 subsystem, allowing a consolidation of LPARs as well as DB2 members, and storage monitoring is also reduced. The net result for this virtual storage constraint relief is reduced cost, improved productivity, easier management, and the ability to scale DB2 much more easily.

DB2 10 increases the limits for CTHREAD, MAXDBAT, IDFORE, IDBACK, MAXOFILR threads. Specifically, the improvement allows a 10 times increase in the number of these threads (meaning 10 times the current supported value at your installation, not necessarily 10 times 2000). So, for example, if in your installation you can support 300-400 concurrently active threads based on your workload, you might now be able to support 3000-4000 concurrently active threads. Table 5-4 summarizes these changes.

Table 5-4 Virtual storage relief allows system parameters with new maximums

Macro	DSNZPARM	New maximum	Description/Comment
DSN6SYSP	CTHREAD	From 2000 to 20000	Specifies the maximum number of allied threads (threads started at the local subsystem) that can be allocated concurrently.
DSN6SYSP	MAXDBAT	From 1999 to 19999	Specifies the maximum number of database access threads (DBATs) that can be active concurrently.
DSN6SYSP	IDFORE	From 2000 to 20000	Specifies the maximum number of allied threads (threads started at the local subsystem) that can be allocated concurrently.
DSN6SYSP	IDBACK	From 2000 to 20000	Specifies the maximum number of concurrent connections that are identified to DB2 from batch.
DSN6SYSP	MAXOFILR	From 2000 to 20000	Specifies the maximum number of data sets that can be open concurrently for processing of LOB file references (same as MAX USERS).

Note that when consolidating and increasing such limits, take care in evaluating normal issues such as exceeding I/O, overloading the log, or real memory capacity. Limiting factors now on vertical scalability for the number of threads, thread storage footprint are as follows:

- ▶ Amount of real storage provisioned
- ▶ ESQA/ECSA (31-bit) storage

Keep also in mind the continued value of data sharing for providing unsurpassed availability and scaling.

5.2.3 Storage use measurements

Figure 5-13 shows the comparison of *dynamic SQL workload* and DBM1 below-the-bar storage used on DB2 9 and DB2 10 (but the application is not rebound), and then, it shows DB2 10 with the application rebound. In this workload 360 threads are each executing 10 different SQL statements. The XPROC/SQL and EDM storage values are the actual used amounts. It does not count storage manager and system overhead of fragmentation, and so on. The XPROC/SQL storage in DB2 9 reflects the “local cache” or SQL statement structures for column procedures allocated below-the-bar. In DB2 10, even when running the DB2 9 bound application, significant DBM1 below-the-bar virtual storage savings are made as both the control block structures associated with the SQL statement and other control blocks are allocated above-the-bar.

The AGENT LOCAL storage is the aggregate total of the non-system storage and includes some storage manager overhead as well as unused space due to fragmentation. The STACK USED is the total actual stack in use and does not include allocated but unused blocks. It also includes stack storage currently being used by system tasks. The large AGENT LOCAL storage in DB2 9 is affected by the application process doing full PREPAREs. In DB2 10, the storage associated with the full PREPAREs is mostly allocated above-the-bar so we do not see it. See Figure 5-13.

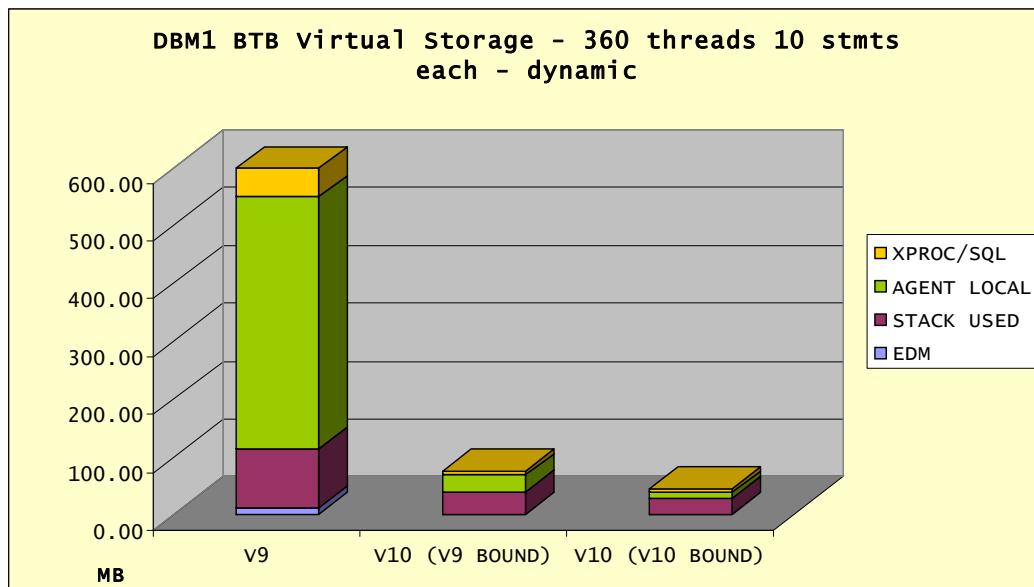


Figure 5-13 DBM1 storage below-the-bar for our dynamic SQL workload

Table 5-5 shows the numbers for Figure 5-13.

Table 5-5 DBM1 Storage below-the-bar for our dynamic SQL workload - numbers

Dynamic SQL	EDM	Stack used	Agent local	XPROC/SQL	Total	% delta
DB2 9	11.25	101.45	436.08	49.94	598.72	
DB2 10 (DB2 9 bound)	0.00	38.34	28.99	6.90	74.23	-87.60
DB2 10 (DB2 10 bound)	0.00	28.14	9.76	6.91	44.81	-92.52

Figure 5-14 shows the comparison of static SQL workload and DBM1 below-the-bar storage used on DB2 9 and DB2 10 (but the application is not rebound), and then when the application is rebound on DB2 10. In this workload the application is almost identical to the dynamic SQL application in the previous chart but uses static SQL instead.

For static SQL applications, the plan and package control blocks and SQL statement structures are created at BIND and can be of significant size. When running a DB2 9 bound application on DB2 10, the EDM equivalent storage is now allocated in the individual thread storage pools. In addition, some control blocks bound in the previous released are ‘puffed’ and allocated in this storage also so the interfaces are compatible with DB2 10.

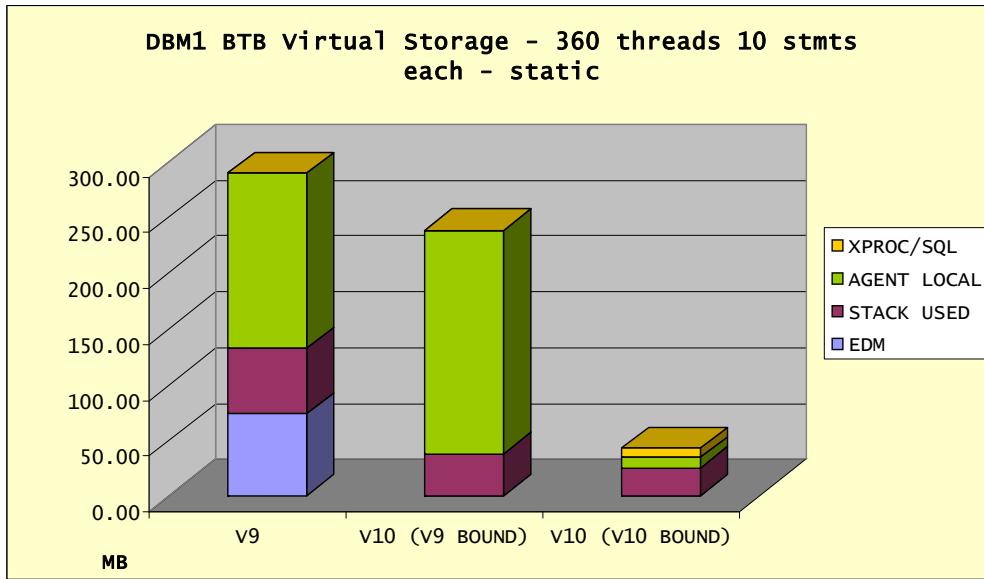


Figure 5-14 DBM1 Storage below-the-bar for our static SQL workload

Table 5-6 shows the numbers for Figure 5-14.

Table 5-6 DBM1 Storage below-the-bar for our static SQL workload - numbers

DYNAMIC SQL	EDM	STACK USED	AGENT LOCAL	XPROC/SQL	TOTAL	% delta
DB2 9	73.85	58.59	156.87	0.00	289.31	
DB2 10 (DB2 9 bound)	0.00	37.88	200.10	0.00	237.98	-17.74
DB2 10 (DB2 10 bound)	0.00	25.34	9.47	8.09	42.90	-85.17

5.2.4 SAP workload

The SAP Sales and Distribution (SD) workload covers a *sell-from-stock* scenario. It includes the creation of a customer order with five line items and the corresponding delivery with subsequent goods movement and invoicing. In this section we show excerpts from the white paper “DB2 10 for z/OS with SAP on IBM System z Performance Report”, which focuses on the performance and scalability provided by DB2 10 in SAP environments, available at this website:

<http://www.ibm.com/support/techdocs/atstrmstr.nsf/WebIndex/WP101845>

This report shows an example of DB2 10 virtual storage relief for a SAP SD workload. There was a huge reduction (up to 90% or more) in virtual storage usage in the DB2 DBM1 address space below the bar on running SAP SD workloads. This is a dramatic improvement in DB2 10 because this was a limitation in DB2 9 for customers running SAP.

Figure 5-15 shows the virtual storage used when running the SAP SD workload with DB2 9 and DB2 10. This test was executed on three different configurations:

- ▶ For the z10 2w (2 CPs) configuration, there were 144 threads connected to a single DB2 subsystem and DB2 10 used only 40 MB of virtual storage in DBM1 compared to 560 MB used with DB2 9.
- ▶ For the z10 4w (4 CPs) configuration, there were 279 threads and DB2 10 used only 54 MB of virtual storage in DBM1 compared to 712 MB with DB2 9.

- For the z10 12w (12 CPs) configuration, there were 409 threads connected to DB2 and the virtual storage usage decreased from 997 MB with DB2 9 to only 63 MB with DB2 10.

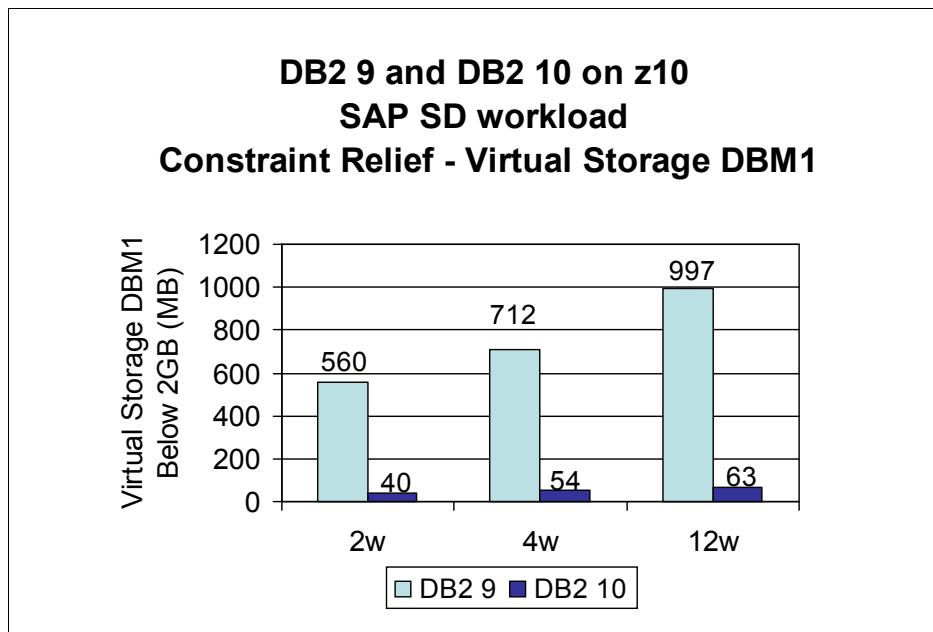


Figure 5-15 DB2 9 and DB2 10 virtual storage usage with SAP SD

DB2 10 constraint relief: More concurrent threads

DB2 10 provides constraint relief in the number of concurrent threads it can support in a single DB2 subsystem. Prior to DB2 10, SAP customers were limited to 400 threads per single DB2 subsystem. To work around this limitation, they needed to use DB2 data sharing. Now, with DB2 10, more concurrent threads can be used within a single DB2 subsystem.

Tests were executed, with DB2 9 and DB2 10 on a z10 12w using the SAP SD workload with 409 threads and 457 threads. This test was able to successfully run 14,400 SD users on a z10 12w using 409 threads with both DB2 9 and DB2 10. However, when increasing the user load to 16,464 SD users, which required additional threads, the test only ran with DB2 10. The details of these runs are documented in Table 5-7. DS are the dialog steps simulating user interaction steps in the SD benchmark.

Table 5-7 Concurrent threads on DB2 9 and DB2 10 with SAP SD

DB2 level	DB2 9	DB2 10	DB2 10
Configuration			
Database server	z10	z10	z10
# of CPs	12	12	12
# of Users	14,400	14,400	16,464
# of DB2 threads	409	409	457
MAXKEEPD in K	8	8	8

DB2 level	DB2 9	DB2 10	DB2 10
Results			
% CPU on z/OS	72.74	61.22	72.14
ETR (DS/sec)	1,421.19	1,422.19	1,623.33
ITR (DS/sec)	1,953.79	2,323.08	2,250.25
Response time (sec)	0.138	0.131	0.148
Virtual storage for DBM1 below 2 GB bar in MB	9997	63	64

These tests show that a single DB2 10 subsystem can support more DB2 threads than DB2 9. Going from 409 threads to 457 threads for this workload is not that significant, but the 457 threads by DB2 10 were limited by the hardware resources available for the test environment from the application side.

To showcase the benefits of DB2 10 as to the number of concurrent threads that it can support in a single DB2 subsystem, a special test was run with 2,505 threads using the SAP SD workload. Special test means that a special SAP profile parameter was used to manage the SAP dispatching of work to its work processes in a round robin fashion.

There were 13,440 SD users on a z10 14w with 2,505 threads connected to a single DB2 10 subsystem. With 2,505 concurrent active threads, the virtual storage used in DBM1 below the 2 GB bar was only 223 MB. The details of this test are documented in Figure 5-16.

- DB2 level	DB2 10
- Database server	z10
- # of CPs	14
- Real storage avail	52 GB
- # of users	13,440
- # of DB2 threads	2,505
- MAXKEEPD	8K
- %CPU on z/OS	51.45%
- ETR (DS/sec)	1,331.40
- ITR (DS/sec)	2,587.76
- Response time (secs)	.099
- DBM1 below 2 GB	223 MB

Figure 5-16 Measurement data - 2,500 threads with DB2 10

DB2 10 constraint relief: Higher MAXKEEPD

SAP applications benefit greatly from using the dynamic statement cache. However, prior to DB2 10, its use was limited because of the virtual storage constraint in the DBM1 address space below the 2 GB bar. With versions of DB2 prior to DB2 10, SAP advised using a starting value of 8,000 for the MAXKEEPD parameter. Then, depending on the virtual storage situation in the DBM1 address space, reduce that value in steps of 2,000 (even down to 0 if needed).

In DB2 10, the local dynamic statement cache is now above the bar in DBM1. Because of this, a larger value can be used for the MAXKEEPD parameter in SAP systems. This means that more prepared statements can be saved past a commit point and fewer “implicit” PREPAREs will be needed. Fewer implicit PREPAREs improve performance.

These tests ran the SAP SD workload on a z10 14w using three different values for MAXKEEPD, 0K, 8K, and 64K (actually, 64K-1 which is 65,535). The results show that as the value of MAXKEEPD is increased, the ITR increases significantly. By increasing the value from 0K to 8K, the ITR increased by 17.5%. The ITR increased by 24% by increasing the value of MAXKEEPD from 8K to 64K. The most dramatic improvement came from increasing the value of MAXKEEPD from 0K to 64K. In this case, the ITR increased by 46%.

These tests results show that the response time is reduced as the value of MAXKEEPD increases. With MAXKEEPD at 0K, the response time is 1.492 seconds. Increasing MAXKEEPD to 8K, reduces the response time to 0.696 seconds. Increasing it to 64K reduces the response time to 0.373 seconds.

As you raise the value of MAXKEEPD, it is important to look at the local statement cache hit ratio. When the local statement cache hit ratio reaches 100%, raising the value of MAXKEEPD will no longer improve performance. A local statement cache hit ratio of 100% means all the SQL statements being executed are found in the local statement cache and no “implicit” PREPAREs are needed. So, in this case, increasing the number of statements that can be kept in the cache will no longer improve performance.

The tests results also show that the virtual storage in DBM1 below the 2 GB bar does not increase as the value of MAXKEEPD is raised. This is great news! And, this is what allows you to use higher values of MAXKEEPD when running with DB2 10.

Figure 5-17 plots the ITR and response times achieved when running the SAP SD workload with DB2 10 with values of MAXKEEPD of 0K, 8K, and 64K. The ITR increases and the response time decreases as the value of MAXKEEPD is raised.

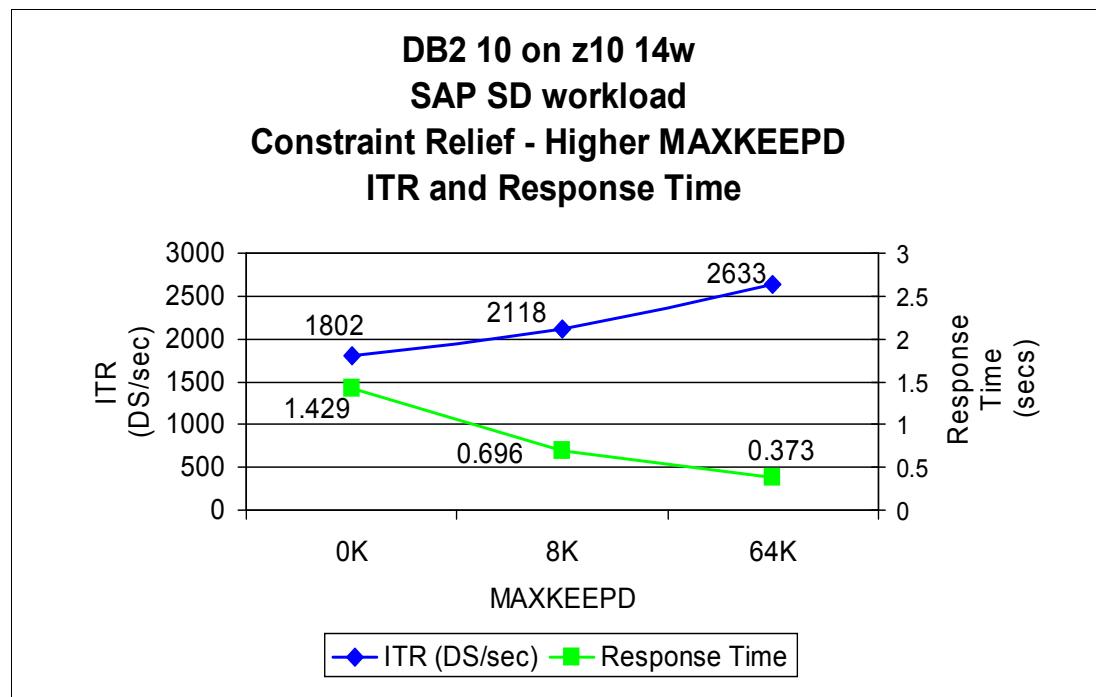


Figure 5-17 DB2 10 ITR and response time when vary MAXKEEP with SAP SD

Figure 5-18 shows how the virtual storage in DBM1 below the 2 GB bar remains relatively constant when running the SAP SD workload with DB2 10 as the value of MAXKEEPD is raised.

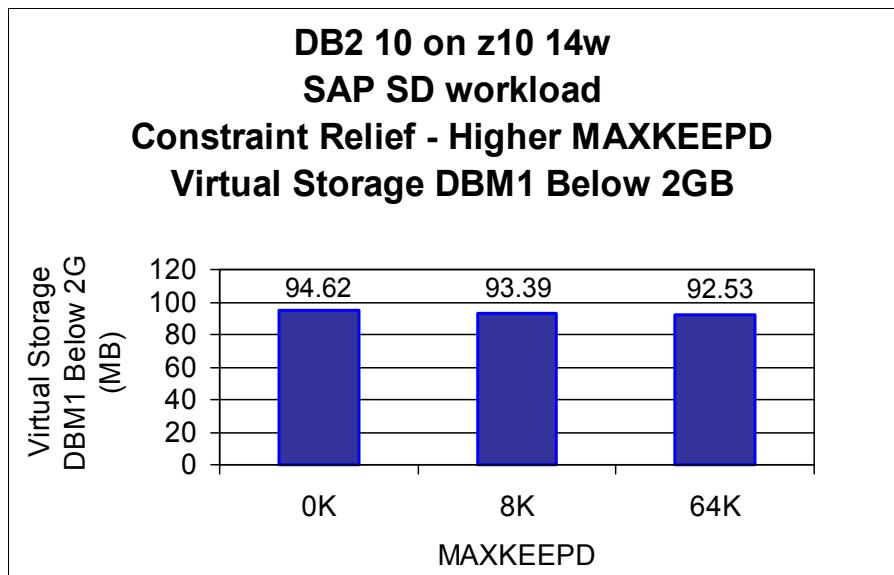


Figure 5-18 DB2 10 virtual storage when varying MAXKEEP with SAP SD

The results show that DB2 10 for z/OS delivers significant performance and scalability improvements compared to DB2 9.

DB2 10 largely eliminates virtual storage as a constraint to thread growth.

5.3 INSERT performance improvements

In this section, we discuss the major improvements that are now available for helping with the insert intensive applications.

5.3.1 Insert performance summary

In Figure 5-19 we first summarize the main enhancements to insert processing with both DB2 9 and DB2 10.

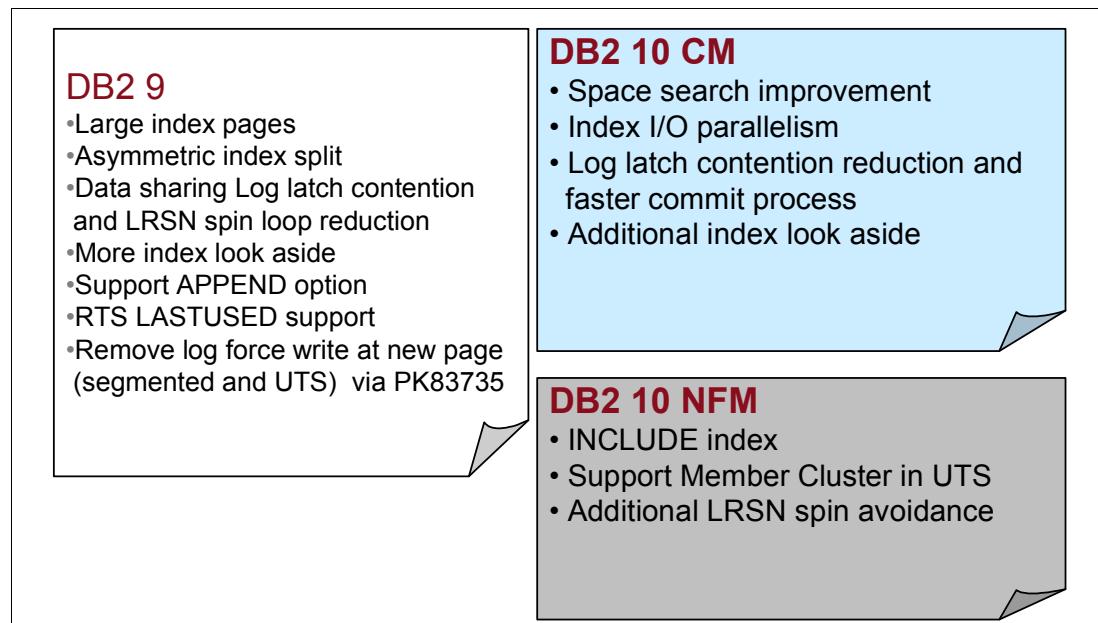


Figure 5-19 Summary of main insert performance improvements

DB2 9 introduced the following improvements for heavy INSERT processing applications:

- ▶ Large index page support:
Index pages larger than 4 KB (enable index compression) reduce the index splits proportionally to the increase in size of the page.
- ▶ Asymmetric index split:
Based on the insert pattern, DB2 splits the index page by choosing from several algorithms. If an ever-increasing sequential insert pattern is detected for an index, DB2 splits index pages asymmetrically using approximately a 90/10 split. If an ever-decreasing sequential insert pattern is detected in an index, DB2 splits index pages asymmetrically using approximately a 10/90 split. If a random insert pattern is detected in an index, DB2 splits index pages with a 50/50 ratio.
- ▶ Data sharing log latch contention and LRSN spin loop reduction:
it allows duplicate LRSN values for consecutive log records for different pages on a given member.
- ▶ More index look aside:
DB2 keeps track of the index value ranges and checks whether the required entry is in the leaf page accessed by the previous call. It also checks against the lowest and highest key of the leaf page. If the entry is found, DB2 can avoid the getpage and traversal of the index tree.

► APPEND=YES:

Requests data rows to be placed into the table by disregarding the clustering during SQL INSERT and online LOAD operations. Rows are appended at the end of the table or partition.

► RTS LASTUSED column:

RTS records the day the index was last used to process an SQL statement. Not used indexes can be identified and DROP'd.

► Remove log force write at new page through PK83735:

Forced log writes are no longer done when inserting into a newly formatted or allocated page for a GBP dependent segmented or universal table space.

For details, see *DB2 9 for z/OS Performance Topics*, SG24-7473.

DB2 10 further improves the performance for heavy INSERT applications, through these features:

► Space search improvement:

DB2 10 improves sequential repetitive inserts into the middle of the table based on the clustering index by choosing the candidate page as the same page where DB2 last found enough free space. See 2.8, "Space search improvement" on page 45.

► I/O parallelism for index updates:

DB2 10 provides the ability to insert into multiple indexes that are defined on the same table in parallel. Index insert I/O parallelism manages concurrent I/O requests on different indexes into the buffer pool in parallel, with the intent of overlapping the synchronous I/O wait time for different indexes on the same table. See 2.7, "I/O parallelism for index updates" on page 43

► Log latch reduction in both data sharing and non data sharing:

DB2 9 log latch reduction in data sharing and DB2 10 reduction in both non data sharing and data sharing. For further information, see 2.2, "Latching contention relief" on page 22.

► Log record sequence number spin avoidance for inserts to the same page:

In DB2 10 NFM, consecutive log records for inserts to the same data page (for instance, when using multi-row access) can have the same LRSN value. If consecutive log records are to the same data page, DB2 no longer needs to "spin" waiting for the LRSN to increment. 2.9, "Log record sequence number spin avoidance for inserts to the same page" on page 46.

► Referential integrity:

When inserting into a dependent table, DB2 must access the parent key for referential constraint checking. DB2 10 reduces the CPU overhead of referential integrity checking by minimizing index probes for parent keys. See 7.2, "Referential integrity checking improvements" on page 202.

► Compression on INSERT:

With DB2 10 NFM, you can turn on compression with ALTER any time, and the compression dictionary is built when you execute the following statements:

- INSERT statements
- MERGE statements
- LOAD SHRLEVEL CHANGE

5.3.2 Insert performance measurements

In this section we show the results of several INSERT workload measurements. Insert workloads are becoming more and more intensive and DB2 10 has improved the insert performance with the new and enhanced features we have described in the previous sections.

Sequential insert: Data sharing and multi-row: MEMBER CLUSTER

In the first scenario we executed sequential key insert to populate 3 tables from JDBC clients in a two way data sharing groups, using multi row insert (num-rows is 100). The table space types are: classic segmented (SEG), partition-by-growth (PBG) and partition-by-growth with member cluster (PBG/MC).

The member cluster option is described in 4.1.5, “MEMBER CLUSTER option available for UTS” on page 80.

The results are shown in Figure 5-20.

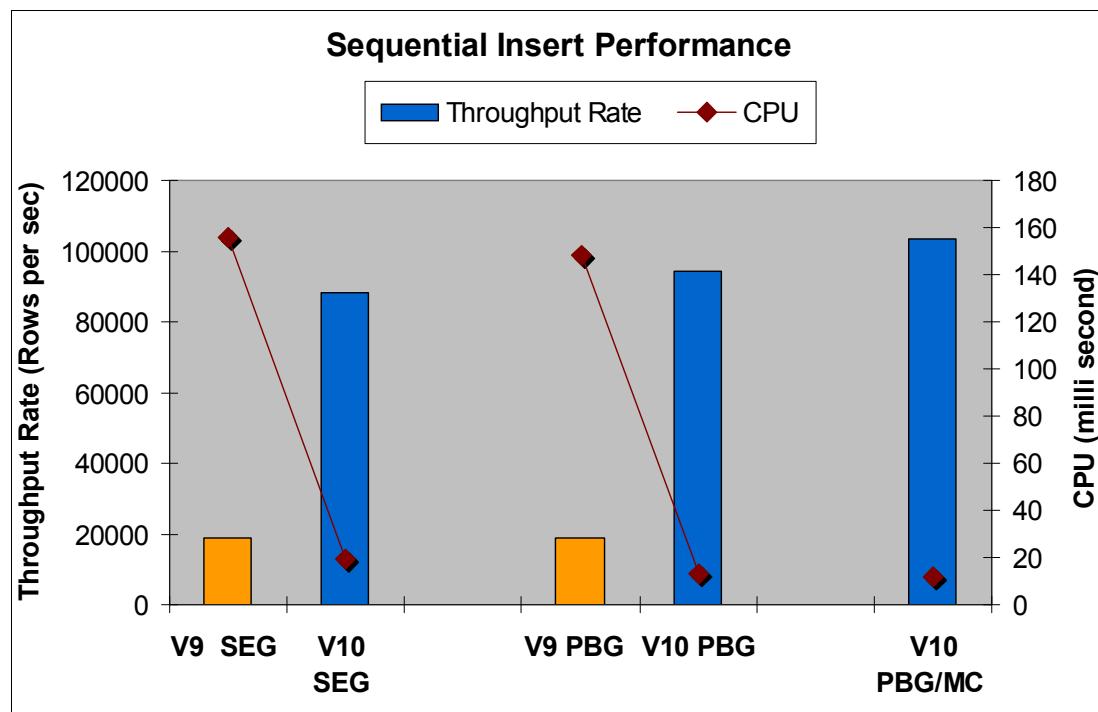


Figure 5-20 Sequential insert performance improvement

The results show that CPU decreases while the throughput rate increases 4 times.

Middle sequential insert: Data sharing and multi-row

In the second scenario we executed middle sequential insert 100,000 times, not multi-row inserts, no logging, on universal table space with 200 byte rows and 8 byte keys.

The results are shown in Figure 5-21.

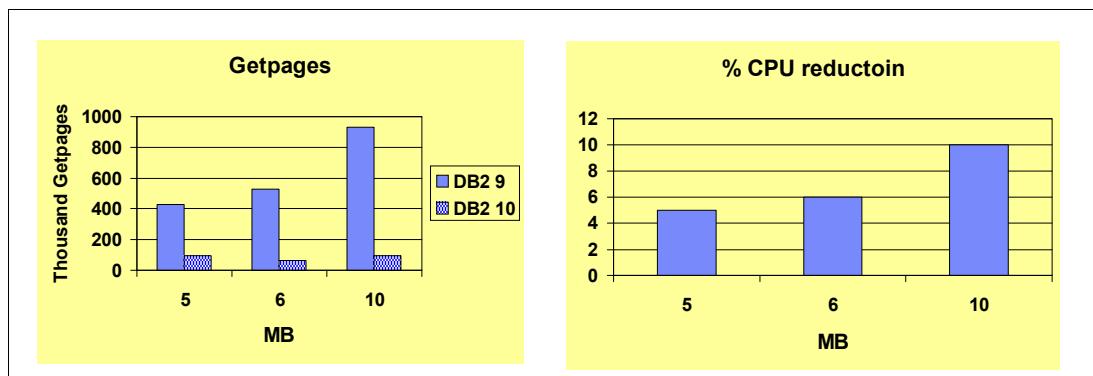


Figure 5-21 Middle-sequential inserts

In this case the getpage savings and consequent CPU reduction was due to avoiding searching for free space. The actual savings will depend on the size of the table space and the amount free space that exists.

Sequential insert: Data sharing and multi-row

The next workload is a high sequential INSERT workload in 2-way data sharing environment with 3 tables with a total of 6 indexes. We show the INSERT performance of DB2 10 relative to DB2 9 of partitioned table space (PTS), segmented (SEG), and UTS (either PBR or PBG) table spaces, with and without member cluster (MC).

A Java application was executed using 240 concurrent threads and multi-row inserts with row set 100, committing every 3 inserts.

A set of charts presents values in percentage change from DB2 9 to DB2 10 for different types of table spaces. Figure 5-22 shows the results.

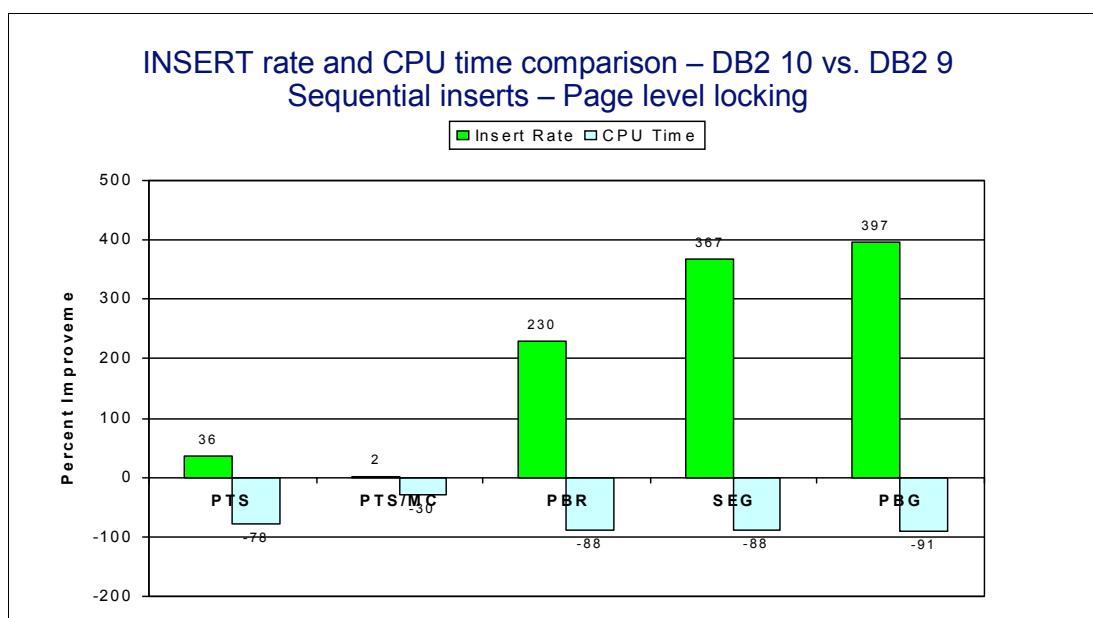


Figure 5-22 Sequential INSERT

The results show that DB2 10 versus DB2 9 insert rate (above the bar) improved up to 400% and CPU time (below the bar) improved up to 90%.

Random insert: Data sharing and multi-row

The last workload is a high random INSERT workload in a 2-way data sharing environment with 3 tables with a total of 6 indexes, to show the INSERT performance of the partitioned table space (PTS), segmented (SEG), and UTS (either PBR or PBG) table spaces.

The last scenario includes a random insert workload that was executed using 200 concurrent threads and single-row inserts. The results are shown in Figure 5-23.

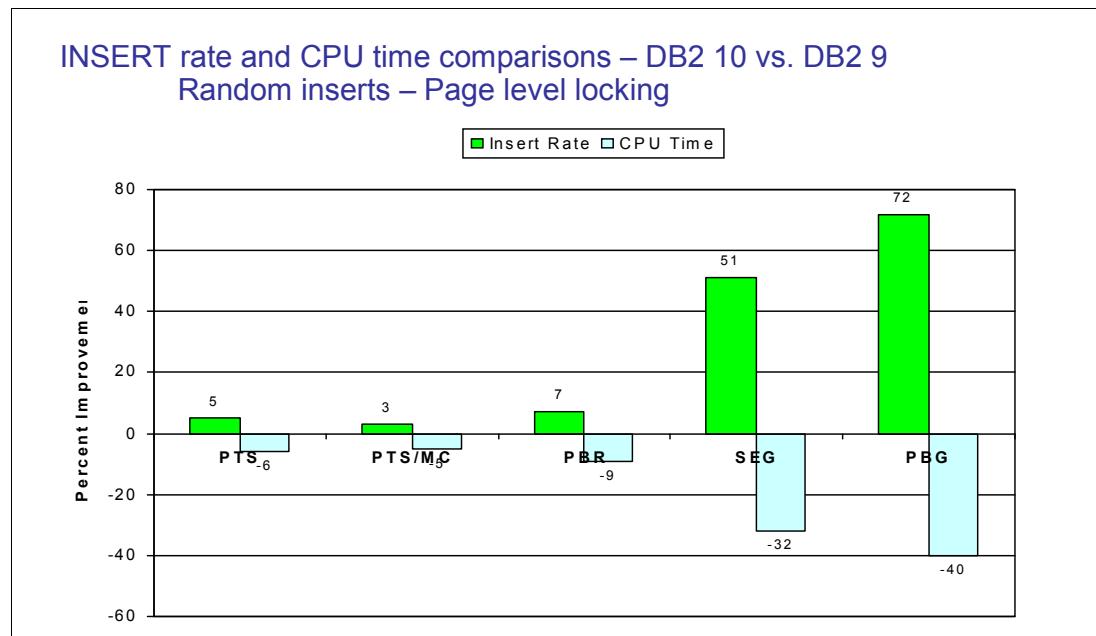


Figure 5-23 Random INSERT

The results show that the DB2 10 versus DB2 9 insert rate improved up to 72% and CPU time improved up to 40%.

More detailed measurements provide a more direct comparison of table space organization performance for sequential insert and random insert, for row level and page level locking in 4.1.6, “UTS workload performance” on page 81.



SQL

DB2 10 for z/OS provides a number of SQL enhancements. The majority of new user features and enhancements to existing SQL functionality in DB2 10 for z/OS are geared towards supporting capabilities that help coding applications and complying with SQL standards that help porting applications across database products and platforms. These enhancements are described in the *DB2 for z/OS Technical Overview*, SG24-7892 and DB2 manuals.

In this chapter we discuss the SQL enhancements that relate to performance improvements. Most of them are changes in the way DB2 executes SQL statements with minimal or no user intervention needed.

In this chapter, we discuss the following topics:

- ▶ IN-list enhancements
- ▶ Range-list index scan
- ▶ Parallelism enhancements
- ▶ Predicate processing enhancements
- ▶ Index probing
- ▶ RID list work file overflow
- ▶ Aggressive merge for views and table expressions
- ▶ Implicit casting extension

6.1 IN-list enhancements

DB2 10 provides a number of improvements to the performance of IN-list processing.

The following sections describe the performance impact of each IN-list enhancement.

- ▶ Matching multiple IN-list predicates
- ▶ Avoiding additional index probing overhead
- ▶ IN-list predicate transitive closure
- ▶ List prefetch access for IN-list

6.1.1 Matching multiple IN-list predicates

DB2 10 removes the restriction that only one IN-list predicate can be a matching predicate. The SQL statement in Example 6-1 shows a SELECT statement with multiple IN-list predicates.

Example 6-1 Matching multiple IN-list predicates

```
SELECT P_SIZE, P_TYPE
FROM   PART
WHERE P_MFGR IN ('MANUFACTURER#3','MANUFACTURER#2')
      AND P_SIZE IN (1,3,5,7,10)
      AND P_TYPE IN ('EC','AN','LA','ST','AB');
```

Index UXP@SZPT has been created on table PART on columns P_SIZE and P_TYPE. Prior to DB2 10, DB2 can only match on one column of index UXP@SZPT. In DB2 10, both IN-lists can be matching predicates. The decision is made based on cost. If DB2 does choose both of the IN-lists as matching predicates, then the values of the IN-list predicates are merged and stored in an IN-list in-memory table. In the case of the foregoing example, the IN-list in-memory table is populated with the following pairs of values for the column pairing P_SIZE, P_TYPE:

```
(1,'EC'),(1,'AN'),(1,'LA'),(1,'ST'),(1,'AB'),
(3,'EC'),(3,'AN'),(3,'LA'),(3,'ST'),(3,'AB'),
(5,'EC'),(5,'AN'),(5,'LA'),(5,'ST'),(5,'AB'),
(7,'EC'),(7,'AN'),(7,'LA'),(7,'ST'),(7,'AB'),
(10,'EC'),(10,'AN'),(10,'LA'),(10,'ST'),(10,'AB')
```

DB2 can then match on columns P_SIZE and P_TYPE instead of just on column P_SIZE.

The access path for using the in-memory table and matching multiple IN-list predicates is shown in Figure 6-1.

QBLOCKNO	PLANNO	METHOD	TNAME	ACCESS TYPE	MATCH COLS	ACCESS NAME	TABLE_TYPE
1	1	0	DSNIN002(01)	IN	0		I
1	2	1	DSNIN003(01)	IN	0		I
1	3	1	PART	I	2	UXP@SZPT	T

Figure 6-1 Explain for matching multiple IN-list predicates

The first two rows in the PLAN_TABLE show a new table type I (column TABLE_TYPE) and a new access type (column ACESSTYPE) of IN. These rows represent the in-memory table access. The naming convention for the in-memory tables is as follows:

- ▶ DSNIN indicates that it relates to IN-list.
- ▶ The number after DSNIN (002 or 003) represents the predicate number.
- ▶ The number in parenthesis represents the query block number.

Note from the Explain data that DB2 is now able to match on both columns of the index. The performance comparison of the one column match in DB2 9 and the two column match in DB2 10 is shown in Table 6-1.

Table 6-1 Matching multiple IN-list predicate performance numbers

Version	Elapsed time (secs)	CPU time (secs)	Index getpages	List prefetch
DB2 9	8.89	0.97	98,368	3,108
DB2 10	2.08	0.02	786	115

The comparison of DB2 9 and 10 numbers shows that matching multiple IN-list predicates result in the following reductions:

- ▶ A 77% reduction in elapsed time
- ▶ A 98% reduction in CPU time
- ▶ A 99.2% reduction in the number of index getpages
- ▶ A 96% reduction in the number of list prefetch requests

Matching multiple IN-list predicates and the use of in-memory work files can provide tremendous benefits when you have multiple IN-list predicates and the columns specified in those IN-list predicates represent two or more leading columns of an index. DB2 10 can combine the IN-list predicates to reduce the number of index getpages and list prefetch operations, resulting in reduced elapsed time and CPU time.

The matching multiple IN-lists enhancement is available in conversion mode.

6.1.2 Avoiding additional index probing overhead

When we look at access paths for queries, usually the higher the number of matching index columns, the better the access path. For example, consider the query in Example 6-2, with a single index on all four predicate columns.

Example 6-2 Avoid additional index probing overhead

```
SELECT L_ORDERKEY,L_SHIPDATE,L_RETURNFLAG, L_SUPPKEY
FROM LINEITEM
WHERE L_ORDERKEY = 5008295
    AND L_SHIPDATE = '1995-12-23'
    AND L_RETURNFLAG = 'A'
    AND L_SUPPKEY IN (9651,9751,9898) ;
```

Prior to DB2 10, the optimizer will choose to match on all four columns of the index, as depicted by MATCHCOLS=4 in the PLAN_TABLE. However, there might be cases where the EQUAL(=) predicates provide strong filtering. In those cases it might be better for the optimizer to match on only 3 columns and then apply index screening to find matches on the fourth column, rather than incur the overhead of additional index probing on the fourth column.

DB2 10 considers the level of filtering in this case and might choose to reduce MATCHCOLS from 4 to 3 and then apply index screening on the fourth column.

The change in access path in DB2 10 results in a reduction in the number of matching columns and a change in the access type from an IN-list index scan to an index access. A test of the query in Example 6-2 on page 155 showed the following results, comparing DB2 10 to DB2 9:

- ▶ MATCHCOLS reduced from 4 to 3
- ▶ ACCESTYPE changed from “N” to “I”
- ▶ About a 15% reduction in CPU time

A reduced number of matching columns for the same query from DB2 9 to DB2 10 does not mean that the access path has degraded. Instead, DB2 10 is able to take advantage of strong filtering provided by EQUAL predicates and avoid the overhead of index probing for IN-list predicates on additional columns of the same index.

The enhancement to avoid additional index probing by reducing the number of matching columns is available in conversion mode.

6.1.3 IN-list predicate transitive closure

Predicate transitive closure has been available for many types of predicates over many different versions of DB2. Starting with DB2 10, predicate transitive closure is now also supported for IN-list predicates. For example, consider the query in Example 6-3, which joins the ORDER and CUSTOMER tables, which have index IXO on column O_CUSTKEY and index IXC on column C_CUSTKEY, respectively.

Example 6-3 Predicate transitive closure for IN-lists

```
SELECT C_CUSTKEY, O_ORDERKEY
FROM   ORDER, CUSTOMER
WHERE  O_CUSTKEY IN (1, 100, 200, 300)
      AND O_CUSTKEY = C_CUSTKEY;
```

In DB2 9, the optimizer is unlikely to consider the CUSTOMER table as the first table accessed, because there is a highly filtering predicate on the ORDER table and the only predicate on the CUSTOMER table is a join predicate.

DB2 10 can apply predicate transitive closure and generate the following predicate:

AND C_CUSTKEY IN (1, 100, 200, 300)

DB2 10 can now consider the CUSTOMER table as the first table accessed if it provides better filtering than the ORDER table.

A test of the query in Example 6-3 resulted in the access path shown in Figure 6-2.

QBLOCKNO	PLANNO	METHOD	TNAME	ACCESS TYPE	MATCH COLS	ACCESS NAME	TABLE_TYPE
1	1	0	CUSTOMER	N	1	IXC	T
1	2	1	ORDER	I	1	IXO	T

Figure 6-2 Explain for IN-list predicate transitive closure

Note that the CUSTOMER table is accessed first, as indicated by a PLANNO value of 1. Because there are no predicates on the CUSTOMER table other than a join predicate, the Explain results show that predicate transitive closure is taking place.

Predicate transitive closure can be applied to IN-lists when the IN-list is coded on one table and the same column is used as a join predicate on another table. DB2 can then take advantage of a lower cost access path by accessing first the table that provides better filtering.

Predicate transitive closure for IN-lists is available in conversion mode.

6.1.4 List prefetch access for IN-list

DB2 10 allows for list prefetch as a valid access path for IN-list predicates. If the IN-list predicate is selected as a matching predicate and list prefetch is chosen, the values for the IN-list predicate are accessed as an in-memory table. For example, consider the query in Example 6-4.

Example 6-4 List prefetch for IN-list predicates

```
SELECT *
FROM T1
WHERE T1.C1 IN (?, ?, ?);
```

DB2 can now use list prefetch to access the index entries for column C1 on table T1.

A test of the query in Example 6-4 resulted in the access path shown in Figure 6-3.

QBLOCKNO	PLANNO	METHOD	TNAME	ACCESS TYPE	MATCH COLS	ACCESS NAME	TABLE_TYPE	PREFETCH
1	1	0	DSNIN001(01)	IN	0		I	
1	2	1	T1	I	1	IX1	T	L

Figure 6-3 Explain for list prefetch usage for IN-list access

The first row shows the access to the in-memory table. The second row shows that list prefetch is used on table T1.

DB2 can now use list prefetch for IN-lists when the matching predicate is transformed into an in-memory table. The optimizer will evaluate the cost of using list prefetch and the cost of not using list prefetch and choose the least cost alternative. List prefetch for IN-list predicates is available in conversion mode.

6.2 Range-list index scan

The range-list index scan feature allows for more efficient access for some applications that need to scroll through data. The format of the query is some variation of the following:

```
SELECT C1, C2, C3, ...
FROM T1
WHERE (C1 = 'XXX' AND C2 > 'YYY')
OR C1 > 'XXX'
ORDER BY C1, C2
```

These types of queries are typical for applications that contain cursor scrolling logic where the returned result set is only part of the complete result set. The query will allow you to select the next set of rows with either of the following characteristics:

- ▶ The same value for the first key column and with values higher than the last returned value for the second key column
- ▶ OR, a value for the first key column that is higher than the last returned value for the first key column

These types of queries (with OR predicates) can suffer from poor performance because DB2 cannot use OR predicates as matching predicates with single index access. The alternative method is to use multi-index access (index ORing), which is not as efficient as single index access. Multi-index access retrieves all RIDs that qualify from each OR condition and then unions the result.

DB2 10 can process these types of queries with a single index access, improving the performance of these types of queries. This type of processing is known as a *range list index scan*, although some documentation also refer to it as *SQL pagination*.

Let us look at a sample query that uses range-list index scan and measure the difference in performance from DB2 9 to DB2 10. Consider the query in Example 6-5. The query returns data from the ORDER table, returning only rows that have a customer key = 500 with an order key less than 8996992 or have a customer key less than 500. We only want 50 rows returned and we want the data sorted in customer key, order key order.

Example 6-5 Range-list index scan

```
SELECT  O_ORDERKEY, O_CUSTKEY  FROM ORDER
      WHERE (O_CUSTKEY  = 500  AND  O_ORDERKEY < 8996992  )
            OR (O_CUSTKEY  < 500  )
GROUP BY O_CUSTKEY, O_ORDERKEY
ORDER BY O_CUSTKEY, O_ORDERKEY
FETCH FIRST 50 ROWS ONLY;
```

In this example, the query is scrolling through the data in descending order. Range-list index scan can be used for scrolling through data in ascending or descending order, as long as there is an index on the columns in the predicates.

The access path for the query in Example 6-5 is shown in Figure 6-4.

QBLOCKNO	PLANNO	TNAME	ACCESSNAME	ACCESSTYPE	MATCHCOLS	MIXOPSEQ
1	1	ORDER	UXO@CKOKODSP	NR	2	1
1	1	ORDER	UXO@CKOKODSP	NR	1	2

Figure 6-4 Explain for range-list index scan

The ACESSTYPE of 'NR' indicates that range-list index scan was chosen by the optimizer. The two rows indicate two probes of the same index are done: the first probe uses two matching columns to satisfy the first predicate; the second probe uses one matching column to satisfy the second predicate. No sort is required for any ORDER BY clause because the column order is the same as the index order.

DB2 9 cannot use a matching index scan and instead has to perform index ORing. DB2 10 can use a matching index scan and list prefetch. The performance comparison of the DB2 9 access path and the DB2 10 access path is shown in Table 6-2.

Table 6-2 Range-list index scan performance numbers

DB2 version	Elapsed time (sec)	CPU time (sec)	Data getpages	Index getpages	Work file getpages
DB2 9	0.13	0.028	5077	40	36
DB2 10	0.05	0.011	50	4	0

The comparison of DB2 9 and DB2 10 numbers shows that the use of range-list index scan results in the following reductions:

- ▶ A 62% reduction in elapsed time.
- ▶ A 61% reduction in CPU time.
- ▶ A 99% reduction in the number of index getpages.
- ▶ A 100% reduction in the number of work file getpages (no sort takes place).

Range-list index scan can provide many performance benefits for applications that use scrolling logic on more than one index column:

- ▶ It is an index access with matching predicates. It can narrow down the search scope compared to table space scan or non-matching index scan.
- ▶ It is a single index access instead of multiple indexes access (index ORing). The index is exploited once.
- ▶ It allows index key ordering to be maintained. If the index satisfies ORDER BY ordering, then a sort can be avoided.
- ▶ The process can be terminated early if only part of the result set is required. The FETCH FIRST n ROWS ONLY clause can be exploited.

These benefits can provide significant elapsed and CPU time savings for applications that use SQL pagination, as shown by the performance measurements in Table 6-2.

Range-list index scan is available in conversion mode.

6.3 Parallelism enhancements

DB2 10 provides a number of improvements to query parallelism:

- ▶ Record range partitioning
- ▶ Straw model for workload distribution
- ▶ Sort merge join improvements
- ▶ Removal of some parallelism restrictions
- ▶ Query parallelism degree change
- ▶ Parallelism enhancements performance summary

The following sections describe the performance impact of each parallelism enhancement.

6.3.1 Record range partitioning

To evaluate parallelism, DB2 chooses key ranges decided at bind time by the optimizer, based on statistics (low2key, high2key, and column cardinality) and the assumption of uniform data distribution within low2key and high2key. This makes DB2 dependent on the availability and accuracy of the statistics. If the statistics are inaccurate or there is data skew or data correlation, the key ranges chosen might not result in an even distribution of the workload among the parallel tasks. The uneven distribution of the workload can result in elongated elapsed times due to one or more of the parallel tasks processing considerably more data than the other tasks.

Record range partitioning differs from key range partitioning in that the partitioning is done at execution time instead of bind time and that partitioning is based on an equal number of records instead of based on keys.

The following key range related re-partitioning restrictions or inefficiencies no longer exist with record range partitioning:

- ▶ Limited number of distinct values for leading columns
- ▶ Data skew or data correlation
- ▶ Lack of accurate statistics

See *DB2 10 for z/OS Technical Overview*, SG24-7892, for details on the differences between key range partitioning and record range partitioning.

The query shown in Example 6-6 resulted in key range partitioning in DB2 9.

Example 6-6 Record range partitioning query

```
SELECT DB2R1.TDWKE,
       DB2R1.NUM_LT,
       SUM(QTY_OH_STR_INV)
  FROM SHOTSTRWK DB2R1,
       DL5WDATE DB2R2
 WHERE DB2R1.TDWKE = DB2R2.TDLST5WKE
   AND DB2R2.TDWKE IN ('03/24/2007')
 GROUP BY DB2R1.TDWKE, DB2R1.NUM_LT ;
```

The access path for key range partitioning in DB2 9 is shown in Figure 6-5.

	QUERYNO	QB#	PL#	JT	AT	TNAME	CORNM	
1	100	1	1		I	DL5WDATE	DB2R2	
2	100	1	2		I	SHOTSTRWK	DB2R1	
3	100	1	3				?	
	ACCESSNAME	TT	T#	METH	MATC	MJC	PF	
1	DL5WDATE	T	2	0	1			
2	DLSTWK	T	1	1	1		S	
3		?	0	3	0			
	SCN_JUOG	ACC_DEG	ACC_PGR	J_DEG	J_PGR	SC_PGR	SN_PGR	P_MODE
1		8	1	?	?	?	?	C
2	NNNN-NNNN	8	1	8	1	?	?	C
3	NNNY	?	?	?	?	1	?	?

Figure 6-5 Explain for key range partitioning in DB2 9

The same query results in record range partitioning in DB2 10.

The access path for record range partitioning in DB2 10 is shown in Figure 6-6.

	QUERYNO	QB#	PL#	JT	AT	TNAME	CORNM	
1	100	1	1		I	DL5WDATE	DB2R2	
2	100	1	2		I	SHOTSTRWK	DB2R1	
3	100	1	3				?	
	ACCESSNAME	TT	T#	METH	MATC	MJC	PF	
1	DL5WDATE	T	2	0	1			
2	DLSTWK	T	1	1	1		S	
3		?	0	3	0			
	SCN_JUOG	ACC_DEG	ACC_PGR	J_DEG	J_PGR	SC_PGR	SN_PGR	P_MODE
1		8	1	?	?	?	?	C
2	NNNN-NNNN	5	2	5	2	?	?	C
3	NNNY	?	?	?	?	2	?	?

Figure 6-6 Explain for DB2 10 record range partitioning

The performance comparison of the DB2 9 access path and the DB2 10 access path is shown in Table 6-3.

Table 6-3 Record range partitioning performance numbers

DB2 version	Maximum degree	Number of parallel groups	Elapsed time (sec)	TCB CPU time (sec)
DB2 9	8	1	56.48	69.61
DB2 10	8	2	21.67	71.17
% Difference			-61.6%	+2.2%

You can see that for this test, we saw a 61.6% reduction in elapsed time and a 2.2% increase in TCB CPU time. Many queries in customer workloads have shown significant improvement in elapsed time. The average improvement we have seen is 34%.

Record range partitioning can provide a large performance improvement for queries that have significant data skew and/or join skew. We have observed up to 8 times improvement in query elapsed time for internal tests. the more unbalanced the child tasks are, the more improvement you can expect to see with record range partitioning.

Record range partitioning is available in conversion mode.

6.3.2 Straw model for workload distribution

The straw model is another DB2 10 enhancement that resolves the issue of uneven distribution of work among parallel tasks. Record range partitioning involves sorting data and then splitting up the work evenly, based on key ranges, into a number of tasks defined by the degree of parallelism. The straw model does not sort the data; instead, it breaks up the data into a greater number of key ranges and then executes a number of tasks equal to the degree of parallelism. Each task represents a smaller key range, so the elapsed time for each task is smaller. Each parallel task will continue on the next available range after it finishes the current one. The parallel tasks stop after all the ranges are processed.

The straw model is used instead of record range partitioning when the following criteria are true:

- ▶ The parallel group cost ≥ 1000 AND
- ▶ There is no parallel sort AND
- ▶ The leading table is index access and COLCARD is not too small ($\text{COLCARD} \geq 2 * \text{DEGREE}$) OR the leading table is table space scan and there is a join and the number of pages is not too small ($\# \text{ pages} \geq 2 * \text{degree}$).

Figure 6-7 shows the differences between how record range partitioning divides up the key ranges and how the straw model divides up the key ranges.

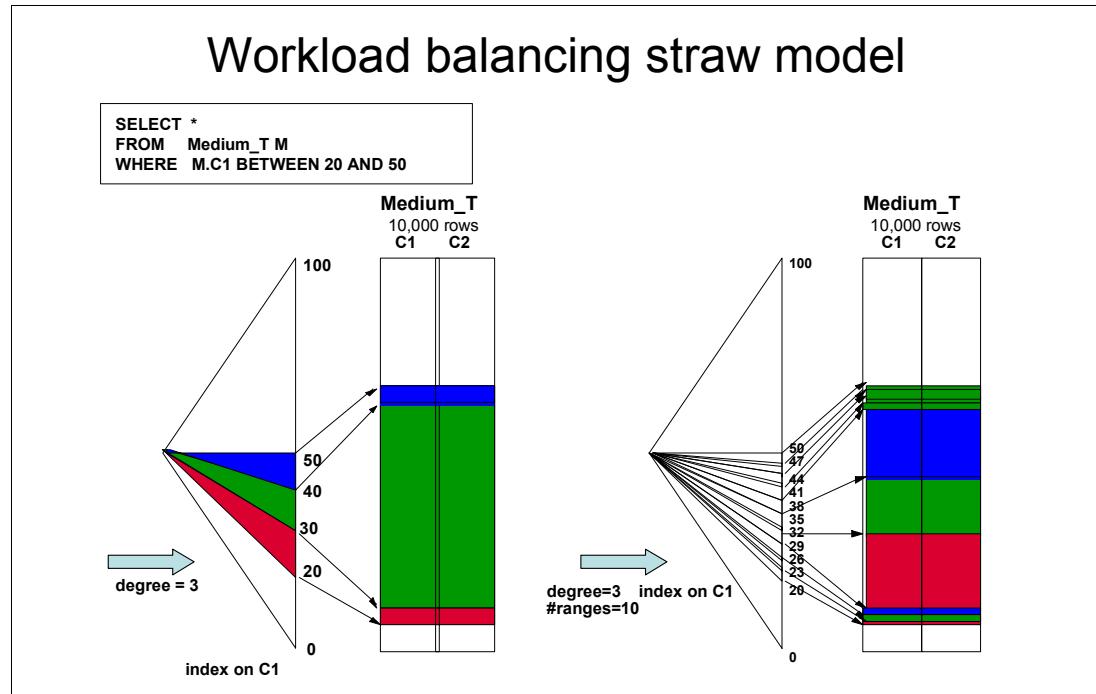


Figure 6-7 Non-straw model versus straw model

The straw model provides performance benefits over record range partitioning for queries that have a skewed distribution of key ranges and therefore a skewed distribution of the workload in the parallel tasks. The most benefits are seen for workloads with unbalanced child tasks.

We have observed up to 49% elapsed time improvement for a join query with skewed distribution. The characteristics of the query are as follows:

- ▶ Two table nested loop join
- ▶ Matching index-only scan
- ▶ One parallel group
- ▶ Degree=8

The CPU overhead for this query was 1.2%. When the query is run without the straw model, the average CPU utilization is about 50%. When the query is run with the straw model, the average CPU utilization is about 99%. This difference shows that the straw model is able to run more work in parallel due to the smaller key ranges being processed and there is no dead time while tasks that complete sooner are waiting for tasks that take longer to complete.

If the straw model is used, column **STRAW_MODEL** in table **DSN_PGROUP_TABLE** will contain a value of 'Y'.

The straw model is a very efficient method to run parallel tasks when there is a skewed distribution of data across key ranges. We saw no big performance difference with or without the straw model when we ran queries and workloads with an even distribution of key ranges.

The straw model method of parallelism is available in conversion mode.

6.3.3 Sort merge join improvements

Prior to DB2 10, when an inner work file is used for a sort merge join, the work file is not partitioned if the number of partitioning keys and sort keys are different or the keys have different data types or lengths. Each parallel child task then has to scan through the whole work file if it is not partitioned, resulting in a negative impact on performance.

DB2 10 addresses this issue by building a sparse index on the inner work file. Each child task will use the sparse index to position the scan, rather than each task having to scan the entire work file. This change in behavior applies only to CP parallelism.

The performance behavior in DB2 10 is different from the traditional behavior for parallel processing when sort merge join is used. The behavioral differences are as follows:

- ▶ The inner work file is no longer partitioned:
 - There is a single shared work file with a sparse index built on it.
 - The sparse index can be hashed, or in-memory, or a physical work file.
- ▶ The sparse index is probed once for each outer table range
- ▶ The restrictions from key range partitioning on both outer and inner sides no longer exist, which results in more parallelism.
- ▶ Parallel sort is not used due to the use of a sparse index.

The benefits of the sort merge join improvements are that the child tasks do not have to scan the entire work file and more parallelism is available.

The sort merge join improvements are available in conversion mode.

6.3.4 Removal of some parallelism restrictions

DB2 10 provides a number of enhancements that remove some restrictions on when parallelism can be used. Prior to DB2 10, DB2 restricted the use of parallelism in the following situations:

- ▶ Parallelism was disabled in the last parallel group of the top query block when multi-row fetch was used.
- ▶ Parallelism was disabled when a parallel group contains a work file from a materialized view or a materialized table expression.
- ▶ Parallelism was disabled when a query block contains OLAP functions.

Parallelism in multi-row fetch

In previous versions of DB2, parallelism was disabled in multi-row fetch for the last parallel group in the top level query block if there is no more table to join after the parallel group and there is no GROUP BY clause or ORDER BY clause. For example, consider the following query:

```
SELECT * FROM CUSTOMER
```

There is no parallel group in the query and there are no table joins. There is no GROUP BY clause. There is no ORDER BY clause. Therefore, parallelism will not be used.

DB2 10 removes this restriction if the cursor is declared as READ ONLY. If the cursor is an ambiguous cursor, then the restriction is not removed.

The query shown in Example 6-7 was run using multi-row fetch.

Example 6-7 Parallelism with multi-row fetch

```
SELECT *
  FROM TABLELI
 WHERE COMMENT = 'GOOD'
 FOR FETCH ONLY ;
```

The performance measurements in DB2 9, when no parallelism can be used for multi-row fetch, and DB2 10, when parallelism can be used, are shown in Table 6-4.

Table 6-4 Multi-row fetch parallelism performance numbers

Version	Elapsed time (sec)	CPU time (sec)	Data getpages	Index getpages
DB2 9	252	40	5.996 million	N/A
DB2 10	37	43	5.996 million	N/A

For both DB2 9 and DB2 10 the access path is a table space scan (ACCESSTYPE=R). However, you can see that there is an 85% reduction in elapsed time when parallelism is used in DB2 10. There is an additional 7.5% CPU cost to use parallelism in this case.

Note that parallelism for multi-row fetch is only supported when the cursor is read-only.

Parallelism when parallel group contains a work file

DB2 generates a temporary work file when a view or table expression is materialized. This type of work file cannot be shared among child tasks in previous versions of DB2; therefore parallelism is disabled.

DB2 10 makes the work file shareable, therefore allowing parallelism. There are some limitations as to when DB2 10 will use parallelism in this case. Parallelism will only be used for CP mode parallelism. In addition, it will not be used for full outer joins.

Parallelism when a query block contains OLAP functions

Prior to DB2 10, parallelism is disabled when a query block contains OLAP functions such as RANK, DENSE_RANK and ROW_NUMBER. In DB2 10, for queries that use OLAP functions, query parallelism can work during the data retrieval if DB2 decides to use parallelism to access the table. However, the processing of the OLAP function is still sequential, meaning there is no query parallelism. Therefore, the OLAP function is processed at the parent side as opposed to at the parallel child.

The removal of these parallelism restrictions is available in conversion mode.

6.3.5 Query parallelism degree change

The PARAMDEG subsystem parameter specifies the maximum degree of parallelism that is to be allowed for a parallel group. When you specify a value for this parameter, you limit the degree of parallelism so that DB2 cannot create too many parallel tasks that use virtual storage. If you specify a value of 0, which is the default value, then DB2 will choose a maximum degree of parallelism that is based on the system configuration.

DB2 10 provides an enhancement to cap the parallelism degree at 2 times the number of CPs when PARAMDEG is set to 0. If PARAMDEG is set to a non-zero value, then it is used to cap the degree of parallelism, unless the degree is provided by an optimization hint.

If PARAMDEG is set to zero, then the difference in behavior between DB2 versions is as follows:

- ▶ Prior to DB2 10, the degree is set to the calculated degree, but is capped at 10 times the number of CPs.
- ▶ In DB2 10, the degree is set to the calculated degree, but is capped at 2 times the number of CPs.

The reason for setting the parallelism degree cap lower in DB2 10 when PARAMDEG is set to 0 is that in DB2 10 there is the potential to have more CPs in one LPAR due to a big reduction in DBM1 virtual storage usage; therefore there is the potential for more parallel tasks if the degree cap is unchanged.

For those customers who use a PARAMDEG value of 0, they might experience degradation in query performance after migration from DB2 9 to DB2 10 due to the possibility of a lower degree of parallelism being chosen for some queries. To get a higher degree of parallelism, those customers can set PARAMDEG to a desired non-zero value as the maximum degree of parallelism.

The value chosen for maximum degree of parallelism is a trade off between the best query response time and the storage used by child tasks running concurrently.

The change in the query parallelism degree cap is available in conversion mode.

6.3.6 Parallelism enhancements performance summary

We ran some performance tests for a set of customer queries to measure the overall impact of the parallelism enhancements and other enhancements in DB2 10. We ran the same set of queries in DB2 9 and in DB2 10. The performance tests showed the following results as an average for all queries:

- ▶ 46% reduction in elapsed time
- ▶ 9% reduction in CPU time
- ▶ 3X reduction in class 3 wait time
- ▶ 15X reduction in DB2 latch contention
- ▶ 5X reduction in sync I/O wait time
- ▶ 21% more parallel groups

Individual queries showed up to a 10 times performance improvement from record range partitioning, straw model, more parallelism, and other enhancements in DB2 10.

6.4 Predicate processing enhancements

In this section we cover two areas of enhancements related to predicate processing:

- ▶ Predicate evaluation enhancement
- ▶ Residual predicate enhancements

6.4.1 Predicate evaluation enhancement

A change was made in DB2 10 to improve the processing for evaluating predicates by using machine code generation for most cases.

The generation of machine code results in much more efficient custom code generated for each predicate. The machine code generation is done at BIND time if possible or dynamically generated at execution time. The LIKE predicate is an example of the type of predicate for which user machine code can be generated, although this is not done if the value in the LIKE predicate is mixed case. Machine code generation is also not done for a descending index.

The following test cases show the benefit of the enhanced predicate evaluation in DB2 10.

Example 6-8 shows a query with three predicates that we used to measure the predicate evaluation enhancement.

Example 6-8 Predicate evaluation enhancements - 3 predicate test

```
SELECT L_PARTKEY
  FROM
    LINEITEM
 WHERE
   L_PARTKEY      > 3000000 AND
   ( L_SUPPKEY    = 7000001      OR
     L_SUPPKEY    = 7000002 );
```

Example 6-9 shows a query with ten predicates that we used to measure the predicate evaluation enhancements.

Example 6-9 Predicate evaluation enhancements - 10 predicate test

```
SELECT L_PARTKEY
  FROM
    LINEITEM
 WHERE
   L_PARTKEY      > 3000000 AND
   ( L_SUPPKEY    = 7000001      OR
     L_SUPPKEY    = 7000002      OR
     L_SUPPKEY    = 7000003      OR
     L_SUPPKEY    = 7000004      OR
     L_SUPPKEY    = 7000005      OR
     L_SUPPKEY    = 7000006      OR
     L_SUPPKEY    = 7000007      OR
     L_SUPPKEY    = 7000008      OR
     L_SUPPKEY    = 7000009      );
```

We ran each of these queries twice: once when there was an index defined that will allow for index access; and once when there was not an appropriate index and the access path will be a table space scan. In both sets of tests the query returned no rows; therefore the cost of the query amounted to the cost to evaluate the predicates.

The CPU savings for our test with index access are shown in Figure 6-8.

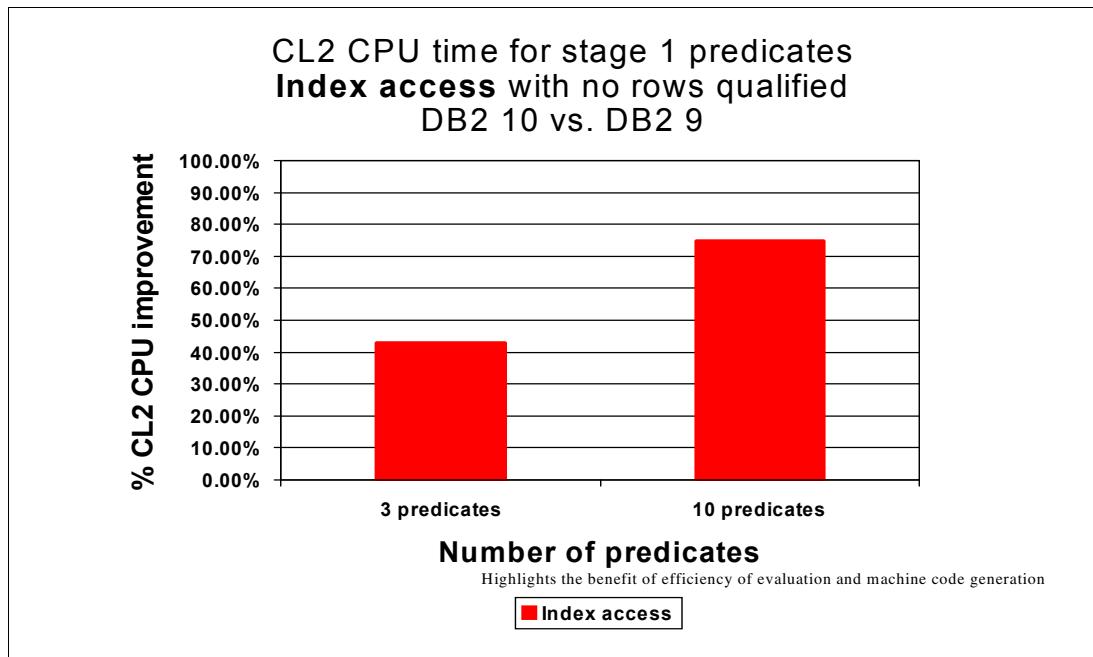


Figure 6-8 Predicate evaluation savings for index access

We saw more than a 40% reduction in class 2 CPU time from DB2 9 to DB2 10 for the case where there were three predicates to be evaluated, and more than a 70% reduction in class 2 CPU time for the case where there were ten predicates to be evaluated. Because no rows were returned in each case, the savings are attributable to the cost to evaluate the predicate. If rows were returned, then there is a cost associated with locking and I/O, so the CPU savings as a percentage of the total cost of the query is less.

The CPU savings for the same test when there is no available index are shown in Figure 6-9.

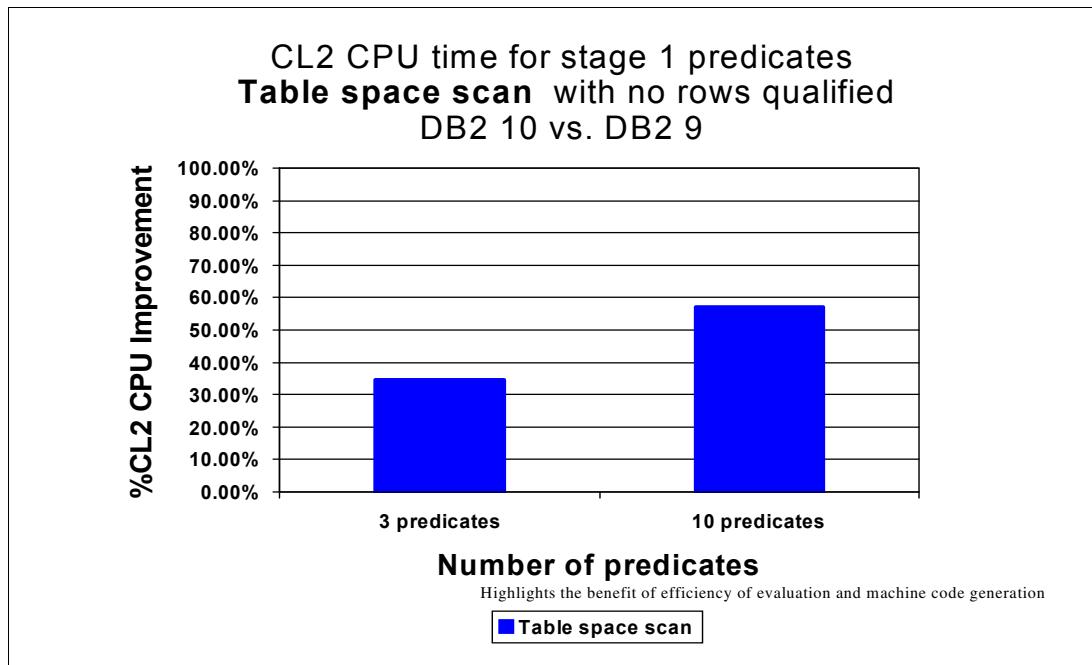


Figure 6-9 Predicate evaluation savings for table space scan access

We see about a 35% reduction in class 2 CPU time from DB2 9 to DB2 10 for the case where there were three predicates to be evaluated and about a 58% reduction in class 2 CPU time for the case where there were ten predicates to be evaluated.

One of the types of predicates that can benefit the most from the enhancement to generate customized machine code is the IN-list predicate. IN-list queries can benefit just from the machine code generation, as well as from efficiencies in reading a large list of values in the IN-list.

We ran three different IN-list queries to show the CPU savings from the machine code generation. In all three queries there were no rows that qualified, so the savings have nothing to do with how quickly DB2 can find a matching value in the list; the savings are purely in the machine code generation.

The first test we ran used an IN-list with 100 values. The query is shown in Example 6-10.

Example 6-10 Machine code generation enhancement - 100 item IN-list test

```
SELECT P_PARTKEY, P_BRAND
  FROM PART WHERE P_SIZE IN (
    200,102,103,104,105,106,107,108,109,110,
    111,112,113,114,115,116,117,118,119,120,
    130,122,123,124,125,126,127,128,129,121,
    131,132,133,134,135,136,137,138,139,140,
    150,142,143,144,145,146,147,148,149,141,
    151,152,153,154,155,156,157,158,159,160,
    161,162,163,164,165,166,167,168,169,170,
    171,172,173,174,175,176,177,178,179,180,
    181,182,183,184,185,186,187,188,189,190,
    191,192,193,194,195,196,197,198,199,101);
```

The second test we ran was using an IN-list with 50 values. The query is shown in Example 6-11.

Example 6-11 Machine code generation enhancement - 50 item IN-list test

```
SELECT P_PARTKEY, P_BRAND
  FROM PART WHERE P_SIZE IN (
    200,102,103,104,105,106,107,108,109,110,
    111,112,113,114,115,116,117,118,119,120,
    130,122,123,124,125,126,127,128,129,121,
    131,132,133,134,135,136,137,138,139,140,
    150,142,143,144,145,146,147,148,149,141);
```

The third test we ran was using an IN-list with 10 values. The query is shown in Example 6-12.

Example 6-12 Machine code generation enhancement - 10 item IN-list test

```
SELECT P_PARTKEY, P_BRAND
  FROM PART WHERE P_SIZE IN (
    200,102,103,104,105,106,107,108,109,110);
```

Figure 6-10 shows the CPU savings in DB2 10 over DB2 9, taking into account just the savings from the customized machine code generation. No rows qualified for either of the three test queries, so there were no benefits gained from ending the IN-list processing as soon as a match was found (no matches were found). Note that the CPU savings are greater for the larger IN-lists, up to 80% with an IN-list of 100 values.

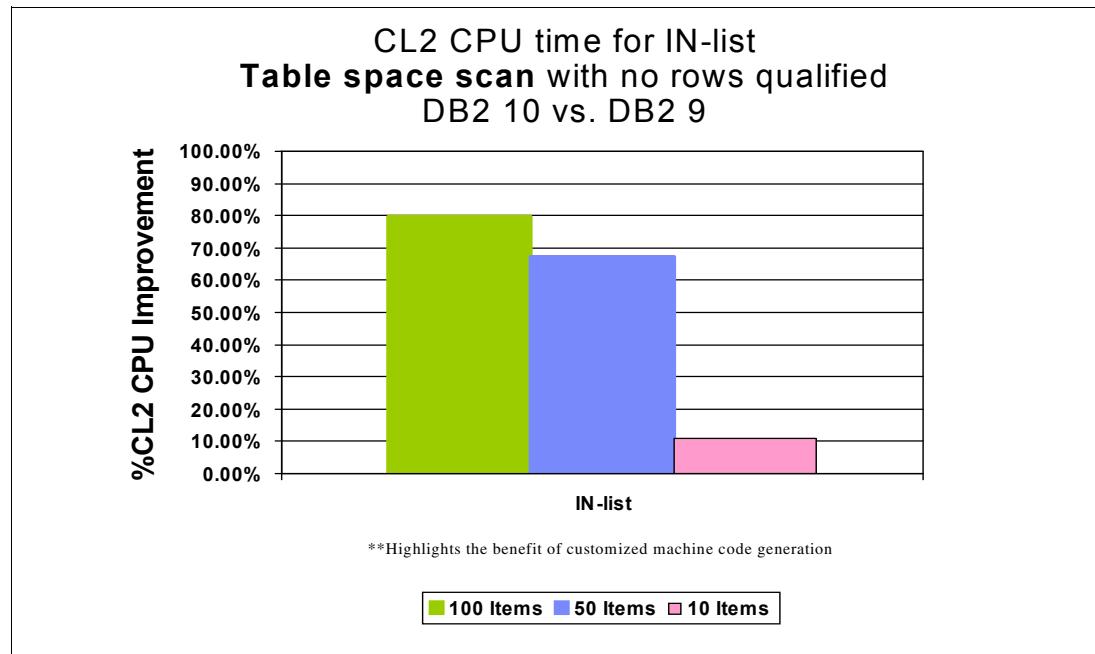


Figure 6-10 Machine code generation savings for IN-list with no qualifying rows

The vast majority of the performance improvements in IN-list processing are due to the machine code generation.

Another of the types of predicates that can greatly benefit from the enhancement to generate customized machine code is the LIKE predicate. We ran a test of the same LIKE predicate in

DB2 9 and DB2 10 and measured the difference. We ran this test using compressed data and using uncompressed data to show the CPU gains in each case. The CPU savings are shown in Figure 6-11. There is a 60% CPU savings for the LIKE predicate in DB2 10 versus DB2 9 when the data is uncompressed. There is a smaller savings, about 22%, when the data is compressed, because the cost to decompress the data is included in the measurement for each version. Again, note that no rows qualify for the LIKE predicate, so the CPU cost savings is just for the predicate evaluation improvements.

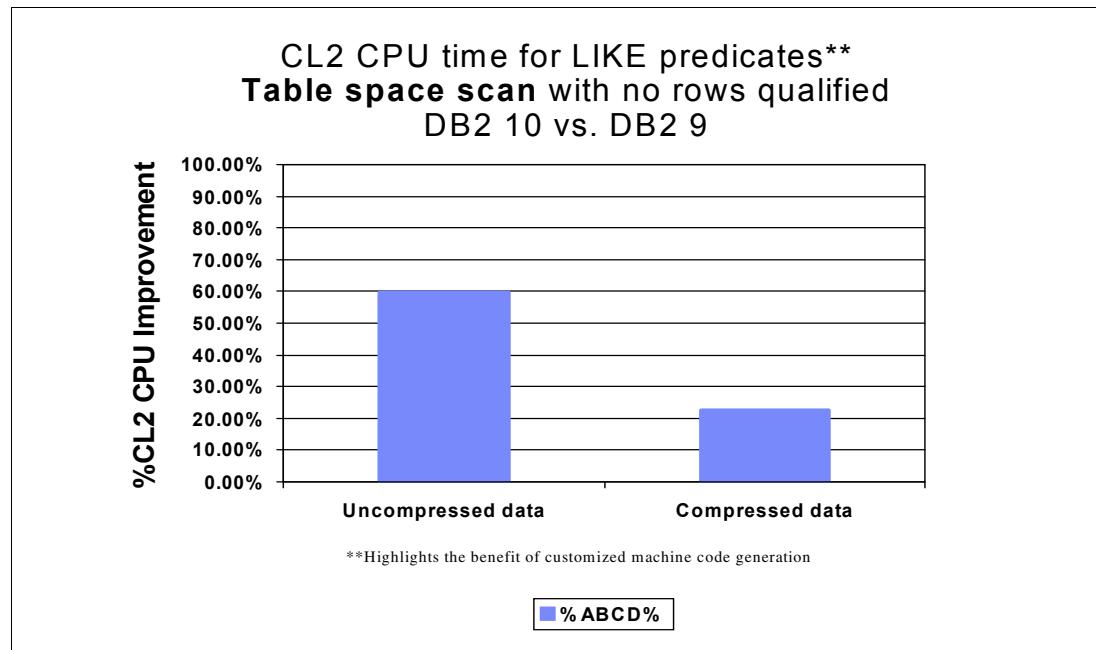


Figure 6-11 Machine code generation savings for LIKE with no qualifying rows

The predicate evaluation enhancements provide improvements in the CPU time associated with evaluating predicates, regardless of how many rows qualify. The improvement has more to do with the complexity of the SQL statement than with the number of rows to be returned. DB2 10 CPU time savings are greater as the number of predicates evaluated increases.

The IN-list processing CPU time improvement is proportional to the number of items in the list. The more items there are in the list, the greater the CPU savings.

For queries with the LIKE predicate, class 2 CPU time improvement is more significant with uncompressed data (60%) than with compressed data (22%).

The predicate evaluation enhancements are available in conversion mode without rebinding, although rebinding does offer some additional benefit, such as possibly generating machine code at bind time instead of dynamically at execution time. If the machine code wasn't generated at bind time, DB2 will generate it dynamically at execution time if the predicate is evaluated often.

6.4.2 Residual predicate enhancements

The DB2 optimizer evaluates predicates in two stages. Predicates that can be applied during the first stage of processing are called Stage 1 predicates. These predicates are also sometimes said to be sargable. Similarly, predicates that cannot be applied until the second stage of processing are called stage 2 predicates, and sometimes described as nonsargable or residual predicates.

The diagram in Figure 6-12 shows the different components of DB2 where predicates can be evaluated. Residual predicates are evaluated in the RDS component, which occurs after predicates are evaluated in the Index Manager (IM) and Data Manager (DM) components. If residual predicates can be pushed down to the other components, then you can improve the performance of your queries.

Residual predicate

▪ What is residual predicate?

- Predicates that cannot be applied until the second stage of predicate processing

▪ Enhancements

- For certain SELECT query processes, DB2 will push down the processing of some residual predicates from RDS component to Data Manager or Index Manager components

▪ Operations / Functions not supported

- UDF specified
- Subquery operations
- Operations related to XML/LOB objects
- SET functions
- Direct row access
- RID-list scan
- Scorable Cursor path
- Workfile scan when retrieving records

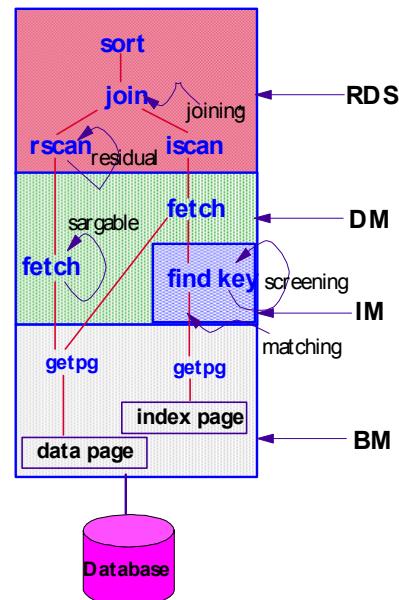


Figure 6-12 Residual predicate enhancements diagram

DB2 10 provides enhancements that push down some of the residual predicates from the RDS component to the Data Manager or Index Manager components in order to evaluate them earlier. For indexed columns, this means the optimizer can apply these stage 2 (residual) predicates as non-matching screening predicates to potentially reduce the number of data rows that need to be accessed. This process eliminates or reduces the amount of data evaluated in stage 2, thus improving query elapsed time and overall query performance.

Certain types of predicates are eligible for this predicate pushdown enhancement. Predicates of the following form are eligible:

- Basic predicate (COL op value)
- BETWEEN predicate
- NULL predicate

In addition, certain types of expressions are also eligible for the predicate pushdown enhancement:

- ▶ Built-in scalar functions (SUBSTR, UPPER, and so on)
- ▶ Built-in operator functions (+, -, /, and so on)
- ▶ Constant
- ▶ Column name
- ▶ Host variable
- ▶ Special register
- ▶ Labeled duration
- ▶ Cast-specification
- ▶ Sequence reference (PREVVAL only)

Columns of all data types are eligible for predicate pushdown, with the exception of LOB and XML columns.

The following test cases show the benefit of the residual predicate enhancements in DB2 10.

Example 6-13 shows a query with three predicates that we used to measure the residual predicate pushdown enhancements.

Example 6-13 Query with three predicates for residual pushdown test

```
SELECT
COUNT(*) AS PART_COUNT
FROM PART
WHERE
    MOD(P_PARTKEY,100) = 0 AND
    ((P_PARTKEY + 10) = 60000010 OR
     (P_PARTKEY + 20) = 60000020);
```

Example 6-14 shows a similar query with ten predicates that we used to measure the residual predicate pushdown enhancements.

Example 6-14 Query with ten predicates for residual pushdown test

```
SELECT
COUNT(*) AS PART_COUNT
FROM PART
WHERE
    MOD(P_PARTKEY,100) = 0 AND
    ((P_PARTKEY + 10) = 60000010 OR
     (P_PARTKEY + 20) = 60000020 OR
     (P_PARTKEY + 30) = 60000030 OR
     (P_PARTKEY + 40) = 60000040 OR
     (P_PARTKEY + 50) = 60000050 OR
     (P_PARTKEY + 60) = 60000060 OR
     (P_PARTKEY + 70) = 60000070 OR
     (P_PARTKEY + 80) = 60000080 OR
     (P_PARTKEY + 90) = 60000090);
```

We ran each of these queries twice: once when there was an index defined that will allow for index access; and once when there was not an appropriate index and the access path will be a table space scan. In both sets of tests the query returned no rows; therefore the cost of the query amounted to the cost to evaluate the predicates.

The CPU savings for our test with index access are shown in Figure 6-13.

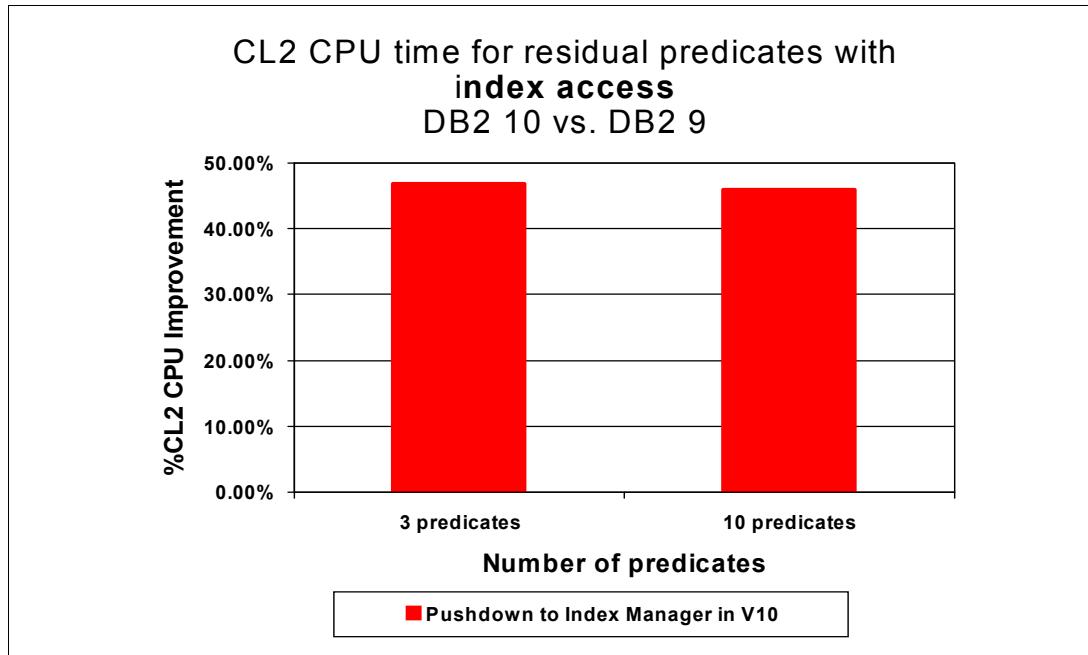


Figure 6-13 Residual predicate pushdown savings for index access

Our tests showed more than a 45% reduction in class 2 CPU time from DB2 9 to DB2 10 for both the three predicate query and the ten predicate query. Because no rows were returned in each case, the savings are attributable to pushing down the predicate evaluation to occur earlier in the process. If rows were returned, then there will be cost associated with returning rows back, so the CPU savings as a percentage of the total cost of the query will be less.

The CPU savings for the same test when there is no available index are shown in Figure 6-14.

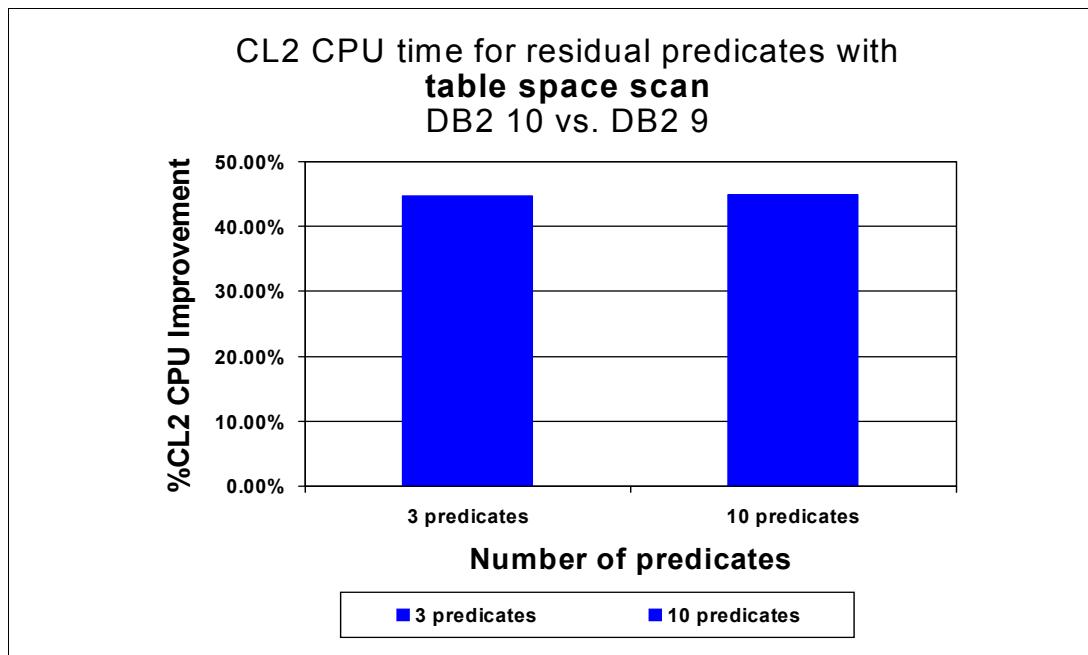


Figure 6-14 Residual predicate pushdown savings for table space scan access

You can see that the savings are fairly similar to the predicate pushdown test for index access.

Because it is unlikely that all your queries will have no rows returned, we ran a number of tests with a varying number of qualifying rows returned. The general form of the query we ran was as follows:

```
SELECT  
  COUNT(*) AS PART_COUNT  
  FROM PART  
 WHERE
```

We ran nine different queries, each with a predicate that qualified a different number of rows. We ran the nine queries twice: once when there was an index to support the predicates; once when there was not. The nine different WHERE clauses and the percentage of rows that will qualify for each are shown in Example 6-15. The PART table was defined as a PARTITION BY GROWTH (PBG) table, with column P_PARTKEY as the partitioning key.

Example 6-15 WHERE clauses for residual predicate pushdown qualifying rows tests

```
Query with 0% rows qualified  
      MOD(P_PARTKEY,60000000) = 0;  
Query with 0.1% (6K) rows qualified  
      MOD(P_PARTKEY,1000) = 0;  
Query with 1% (60K) rows qualified  
      MOD(P_PARTKEY,100) = 0;  
Query with 2% (120K) rows qualified  
      MOD(P_PARTKEY,50) = 0;  
Query with 10%(600K) rows qualified  
      MOD(P_PARTKEY,10) = 0;  
Query with 20% (1.2M) rows qualified  
      MOD(P_PARTKEY,5) = 0;  
Query with 33% (2M) rows qualified  
      MOD(P_PARTKEY,3) = 0;  
Query with 50% (3M) rows qualified  
      MOD(P_PARTKEY,2) = 0;  
Query with 100% (6M) rows qualified  
      MOD(P_PARTKEY,1) = 0;
```

The CPU savings for residual predicate pushdown with a varying number of qualifying rows is shown in Figure 6-15.

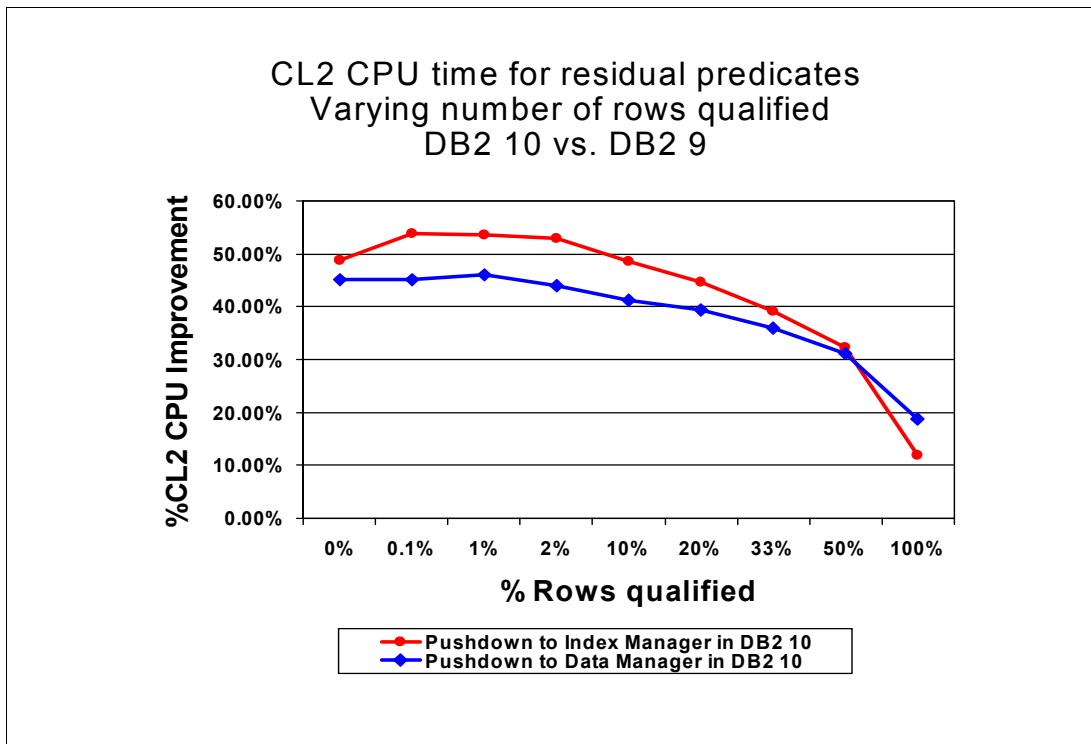


Figure 6-15 Residual predicate pushdown savings for varying number of qualifying rows

The graph shows that the greater percentage savings for the predicate pushdown enhancements are when there are fewer rows that qualify.

The DB2 10 CPU time reduction for the residual predicate pushdown enhancements is mostly from row processing savings because of fewer round trips from the RDS component to the Data Manager component. The DB2 10 CPU improvement increases as the number of qualified rows decreases.

The predicate pushdown enhancements can provide benefits for many queries.

The residual predicate enhancements are available in conversion mode after a rebind.

6.5 Index probing

One of the challenges of query tuning is that some SQL statements are extremely sensitive to statistics and are subject to changing access paths when packages are rebound. The DB2 optimizer can sometimes overestimate the filtering of a predicate, which can occur if a literal value is not known, for example when the predicate is a range predicate, when data is distributed non-uniformly or when host variables or parameter markers are used.

Prior to DB2 10, the preferred solution was to use the REOPT BIND option to allow DB2 to reevaluate access paths based on the actual host variable values. However, this solution is not desirable in all cases because of the re-optimization overhead on each SQL execution.

A method is needed for the optimizer to dynamically discover a better filtering estimate at query execution time for matching predicates that will otherwise have to use a default value for their filter factor estimates.

In DB2 10, the optimizer can use Real Time Statistics (RTS) data and probe the index non leaf pages to come up with better matching predicate filtering estimates. DB2 10 uses index probing in the following situations:

- ▶ When the RUNSTATS utility shows that the table is empty
- ▶ When the RUNSTATS utility shows that there are empty qualified partitions
- ▶ When the catalog statistics have default values
- ▶ When a matching predicate is estimated to qualify zero rows

This new safe query optimization technique, also known as index probing, is only used for matching index predicates with hard coded literals or when REOPT() is used to supply the literals.

Index probing is also used for tables defined as VOLATILE and for tables smaller than the number of pages specified in the NPGTHRSH system parameter, because existing RUNSTATS values are unreliable by definition for these tables.

In order to test the performance of index probing, we performed the following controlled test:

- ▶ Create a table that consists of 8 columns with a single column index on each column
- ▶ Run RUNSTATS on the empty table and associated indexes
- ▶ Insert 2 million rows into the table using random number values for the column contents
- ▶ Insert one row with artificially high values

The INSERT statements we used are shown in Example 6-16.

Example 6-16 INSERT statements for index probing test case

```
INSERT INTO PREDEVALTB (
SELECT
RAND()*2,
RAND()*4,
RAND()*8,
RAND()*16,
RAND()*32,
RAND()*64,
RAND()*128,
RAND()*256 FROM PREDEVALTB);
COMMIT;

INSERT INTO PREDEVALTB
VALUES(9999,9999,9999,9999,9999,9999,9999,9999);
```

Table 6-5 shows the values contained in each of the eight columns, as well as what the COLCARDF values will be if we had run RUNSTATS after inserting the data.

Table 6-5 Distribution of column values for index probing test

Column name	Column values	COLCARDF estimate
COL1	0,1,9999	3
COL2	0,1,2,3,9999	5
COL3	0,1,2,3,4,5,6,7,9999	9
COL4	0-15,9999	17
COL5	0-31,9999	33

Column name	Column values	COLCARD estimate
COL6	0-63,9999	65
COL7	0-127,9999	129
COL8	0-255,9999	257

We ran the same set of steps in DB2 9 and DB2 10. We then ran a number of queries with various types of predicates to see what choices the optimizer will make in DB2 9 and DB2 10 given that the optimizer thinks the table is empty. The table was not defined as VOLATILE. The Explain results are shown in Figure 6-16.

QNO	PREDICATE	V9		V10	
		Access Path Selected	Optimal?	Access Path Selected	Optimal?
10	COL > 8	R	N	INTIX1	Y
11	COL > 16	R	N	INTIX1	Y
12	COL > 9000	R	N	INTIX1	Y
13	COL = 9999	R	N	INTIX1	Y
20	COL > 0	R	Y	R	Y
30	COL = 0	R	N	INTIX8	Y
35	COL = 1	R	N	INTIX8	Y
40	COL = 2	R	N	INTIX1	Y
45	COL IN (1,2,4,8,16,32,64,128)	N (INTIX8)	Y	N (INTIX1)	N
50	COL IS NULL	R	N	INTIX1	Y
60	COL IS NOT NULL	I (INTIX8)	Y	INTIX1	Y
70	COL BETWEEN 0 AND 64	R	N	INTIX8	Y
80	COL = 64 (OR)	R	N	INTIX1 (non matching)	N

• Predicates were written as ‘WHERE COL1 op lit AND COL2 op lit AND COL3 op lit ...’
 • with the exception of QNO 80 which used OR’s.

Figure 6-16 Explain results for index probing (non VOLATILE)

Note that in DB2 9 the access path is a table space scan for 11 out of the 13 queries. The column with the heading of “Optimal?” describes whether the access path chosen is the optimal one. In DB2 9, the optimal access path is chosen for only 3 of the 13 queries. In DB2 10, the optimal access path is chosen for 11 of the 13 queries.

We then ran the same set of tests, but using a VOLATILE table this time. The Explain results are shown in Figure 6-17.

QNO	PREDICATE	Changed DDL to VOLATILE		Changed DDL to VOLATILE	
		Access Path Selected	Optimal?	Access Path Selected	Optimal?
10	COL > 8	INTIX8	N	INTIX1	Y
11	COL > 16	INTIX8	N	INTIX1	Y
12	COL > 9000	INTIX8	N	INTIX1	Y
13	COL = 9999	INTIX8	N	INTIX1	Y
20	COL > 0	INTIX8	N	INTIX1	Y
30	COL = 0	INTIX8	Y	INTIX8	Y
35	COL = 1	INTIX8	Y	INTIX8	Y
40	COL = 2	INTIX8	N	INTIX1	Y
45	COL IN (1,2,4,8,16,32,64,128)	INTIX8	Y	N (INTIX8)	Y
50	COL IS NULL	INTIX8	N	INTIX1	Y
60	COL IS NOT NULL	INTIX8	N	INTIX1	Y
70	COL BETWEEN 0 AND 64	INTIX8	N	INTIX1	Y
80	COL = 64 (OR)	INTIX8 (non matching)	N	INTIX1 (non matching)	N

•Predicates were written as ‘WHERE COL1 op lit AND COL2 op lit AND COL3 op lit ...’
•with the exception of QNO 80 which used OR’s.

Figure 6-17 Explain results for index probing (VOLATILE)

Note that DB2 10 now provides the optimal access path for 12 of the 13 queries, while DB2 9 still only provides the optimal access path for 3 of the 13 queries.

The comparison of the access paths of DB2 9 and DB2 10 show that “Safer Optimization” was chosen for several cases where we were previously basing our selectivity estimates on unreliable information. Index probing is used in some specific cases where we have good reason to not be confident in the filter factor estimate. In these cases, we can now use RTS and/or index probing to come up with a better, more accurate estimate in DB2 10.

Index probing provides improved insurance against disastrous changes in access paths, therefore resulting in better, more stable performance.

Note that index probing is always enabled and cannot be disabled. Index probing is available in conversion mode.

6.6 RID list work file overflow

DB2 10 provides new techniques to better manage the RID pool and overcome its limits. DB2 10 allows an access plan to overflow to work file and continue processing RIDs even when one of the RID thresholds are encountered at run time.

At run time, if the RID pool is filled up, then RIDs overflow to the work file and continue processing with 32 KB sized records, where each record holds the RIDs from one 32 KB RID block. Thus, RID access rarely needs to fall back to a table space scan (as with DB2 9). In some cases, you might see work file use increase as a result. Large memory with a large

work file buffer pool avoids I/O for the RID blocks. Even if there is not enough memory to avoid work file I/O, work file I/O is far better than paging I/O for an oversized RID pool.

In DB2 9, access reverts to a table space scan if the maximum supported number of RIDs (approximately 26,552,680 RIDs) is exceeded. DB2 10 lifts this restriction but only for those RID lists that are stored in a work file. Thus, DB2 first attempts to use work file storage and falls back to table space scan only if DB2 cannot use work file storage *and* the RID list in the work file successfully.

DB2 9 and DB2 10 guarantee that a request for 6524 RIDs or less (one RID block) are satisfied. In prior versions, if the RID pool is full and a small query needs just a single RID pool block, the RID pool requests fails and the query reverts to a table space scan. As a result, a transactional query that normally runs in less than a second now runs minutes or hours due to the table space scan. DB2 has eliminated this problem by guaranteeing each request a single RID block.

As expected, there is some overhead associated with overflowing RID requests to a work file. The following test case shows the overhead. The test used the following criteria:

- ▶ Query using a single index and list prefetch access
- ▶ 15 million RIDs are requested
- ▶ Test #1 used a 1 GB RID pool
- ▶ Test #2 used a 4 MB RID pool
- ▶ No overflow was experienced for the 1 GB RID pool test
- ▶ RID pool overflow occurred at 800,000 RIDs for the 4 MB RID pool
- ▶ 95% of the RID requests are using the work file for test #2

The class 1 and class 2 elapsed and CPU times measured for these two tests are shown in Figure 6-18.

POOL= 1 GB (no overflow)		
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	25:21.6656	10:47.6010
CP CPU TIME	21:19.7364	6:38.01857
POOL= 4 MB (overflow at 800,000 RIDs)		
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)
-----	-----	-----
ELAPSED TIME	26:10.0803	11:35.0692
CP CPU TIME	21:34.2516	6:51.79979

Figure 6-18 RID pool overflow performance numbers

The measurements show a 3% increase in CPU time and a 7% increase in elapsed time. The increases in CPU and elapsed time can be larger if you have a constrained work file buffer pool, but will still be less than the cost of falling back to a table space scan.

As shown in the performance measurements, there is some overhead associated with overflowing RIDs to a work file. However, consider what the overhead will be in the case of a RID pool failure that resulted in falling back to a table space scan. Note also that the work file is not used for hybrid joins.

The RID pool overflow to a work file does not impact access path selection. The overflow to a work file is a run time decision.

DB2 can still fall back to a table space scan if there is not enough work file storage for RID requests. The maximum amount of work file storage that can be used for RID list processing is controlled by DB2 system parameter MAXTEMPS_RIDS.

Overflowing RID list processing to a work file does not eliminate the need to perform capacity planning for your RID pool. The performance measurements show that it is still less costly to access RIDs from the RID pool than it is to access RIDs from a work file. If you have the need to sort large RID lists and you have the real storage to support a large RID pool (larger than the default of 400,000 KB in DB2 10), then it is still more efficient to access RIDs in the RID pool rather than overflow to a work file.

However, be aware that the RID pool size is one of the considerations for access path selection; changing the RID pool size can result in the optimizer choosing RID list processing where it wasn't considered before. DB2 chooses the access path based on cost. A larger RID pool might make RID list processing the least costly alternative in cases where it wasn't previously.

Because RID list processing for hybrid joins cannot overflow to a work file, a large RID pool can provide some extra insurance against a small number of hybrid joins consuming your entire RID pool and causing all shorter running RID list access to overflow to a work file.

The RID pool work file overflow enhancement is available in conversion mode. A rebind of applications is not required; however, we advise that you rebind or bind applications to reset the RID thresholds that are stored with the package.

6.7 Aggressive merge for views and table expressions

In prior versions of DB2, qualifying rows from views and table expressions frequently had to be materialized into a work file for additional processing by a query. The process of physical materialization can add overhead to the processing of a query due to the following limitations:

- ▶ It limits the number of join sequences that can be considered.
- ▶ It can limit the ability to apply predicates early in the processing sequence.
- ▶ The join predicates on materialization work files are not indexable.

Because of these limitations, avoiding materialization to a work file is desirable. DB2 10 provides enhancements to avoid materialization to a work file for views and table expressions, especially when they are involved in outer joins.

6.7.1 Correlated table expression

Example 6-17 shows a query that uses a correlated table expression. The table expression is materialized into a work file in DB2 9 while, in DB2 10, no materialization is necessary.

Example 6-17 Aggressive merge for correlated table expression

```
SELECT *
FROM T1,
TABLE(
    SELECT T1.C2 FROM T1 AS T2
    WHERE T1.C1 = T2.C1
) AS X;
```

The DB2 9 Explain output for the query in Example 6-17 on page 181 is shown in Figure 6-19.

	QUERYNO	QB#	PL#	JT	AT	TNAME	CORNM
1_1	100	1	1		R	T1	?
2_1	100	1	2		R	X	?
3_1	100	2	1		I	T1	T2
+							
	ACCESSNAME	TT	T#	METH	MATC	MJC	PF
		T	1	0	0	S	
		W	3	1	0	S	
x1		T	2	0	1		

Figure 6-19 Explain for DB2 9 correlated table expression query

Note that a work file is used, as denoted by a 'W' in column TABLE_TYPE (denoted as column TT above). DB2 9 has to materialize table expression X. There are two query blocks (QB#) needed to process the query. Materialization of the work file results in 400 work file getpages.

The DB2 10 Explain output for the query in Example 6-17 on page 181 is shown in Figure 6-20.

	QUERYNO	QB#	PL#	JT	AT	TNAME	CORNM
1_1	100	1	1		I	T1	?
2_1	100	1	2		I	T1	T2
+							
	ACCESSNAME	TT	T#	METH	MATC	MJC	PF
x1		T	1	0	0		
x1		T	2	1	1		

Figure 6-20 Explain for DB2 10 correlated table expression query

Note that no work file is used, because the table expression is merged. There is only one query block (QB#) after the merge. There are no work file getpages because there is no materialization.

The performance measurements for the merge of the correlated table expression are shown in Figure 6-21.

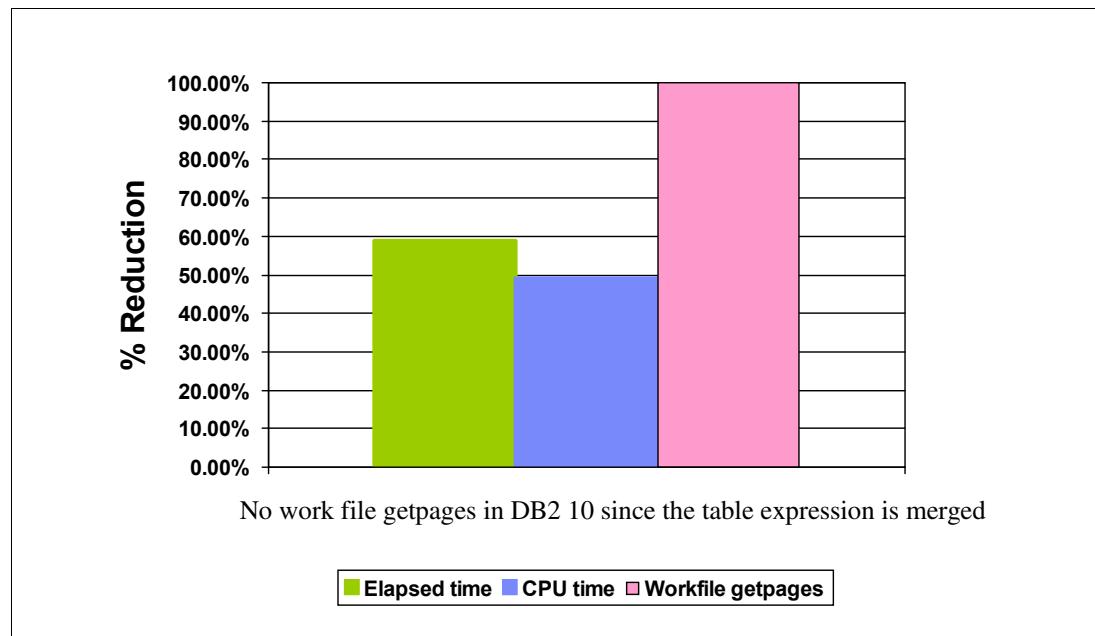


Figure 6-21 Performance for merge of correlated table expression

The aggressive merge of the correlated table expression results in a 59% reduction in elapsed time, a 49% reduction in CPU time, and a 100% reduction in work file getpages.

6.7.2 Table expression on preserved side of outer join

Example 6-18 shows a query that uses two work files in DB2 9, one for table expression B and one for table expression C. One of those table expressions (B) is on the preserved side of a LEFT OUTER JOIN. The preserved side is the side of an outer join that is not replaced with NULL values when there is no match on the join criteria.

Example 6-18 Aggressive merge for table expression on preserved side of outer join

```
SELECT B.C1, B.C2, C.C1, C.C2
FROM T1,
     (SELECT CASE WHEN C1='211' THEN C1 ELSE NULL END AS C1,C2
      FROM T2
     ) B
LEFT OUTER JOIN
     (SELECT CASE WHEN C1='311' THEN C1 ELSE NULL END AS C1,C2
      FROM T3
     ) C
ON   B.C2 = C.C2
WHERE T1.C2 = B.C2;
```

The view or table expression on the preserved side of an outer join can be merged if the following conditions are true:

- ▶ A CASE, VALUE, COALESCE, NULLIF, or IFNULL expression is used.
- ▶ The expression is not used in a predicate after the merge.

Now let us look at the performance of the merge of the table expression on the preserved side of an outer join, as shown in the query in Example 6-18.

The DB2 9 Explain output for the query in Example 6-18 on page 183 is shown in Figure 6-22.

QUERYNO	QB#	PL#	JT	AT	TNAME	CORNM
1	100	1	1	R B	?	
2	100	1	2	I T1	?	
3	100	1	3	L R C	?	
4	100	3	1	I T2	?	
5	100	4	1	I T3	?	
ACCESSNAME	TT	T#	METH	MATC	MJC	PF
x1	W 3	0	0	S		
x1	T 1	1	0			
x2	W 5	1	0			
x2	T 2	0	0			
x3	T 4	0	0			

Figure 6-22 Explain for DB2 9 table expression on preserved outer join query

Note that two work files are used, as denoted by a 'W' in column TABLE_TYPE (denoted as column TT above). DB2 9 has to materialize both table expressions. The query has three query blocks. Materialization of the two work files results in 80 work file getpages.

The same query is Explained in DB2 10. The results are shown in Figure 6-23.

QUERYNO	QB#	PL#	JT	AT	TNAME	CORNM
1	100	1	1	I T2	?	
2	100	1	2	I T1	?	
3	100	1	3	L R C	?	
4	100	4	1	I T3	?	
ACCESSNAME	TT	T#	METH	MATC	MJC	PF
x2	T 2	0	0			
x1	T 1	1	0			
x3	W 5	1	0			
x3	T 4	0	0			

Figure 6-23 Explain for DB2 10 table expression on preserved outer join query

Note that only one work file is used for the same query in DB2 10, as denoted by a ‘W’ in column TABLE_TYPE (denoted as column TT above). The number of work file getpages is cut in half in DB2 10 compared to DB2 9, due to less materialization in DB2 10.

The performance measurements for the merge of the table expression on the preserved side of the outer join are shown in Figure 6-24.

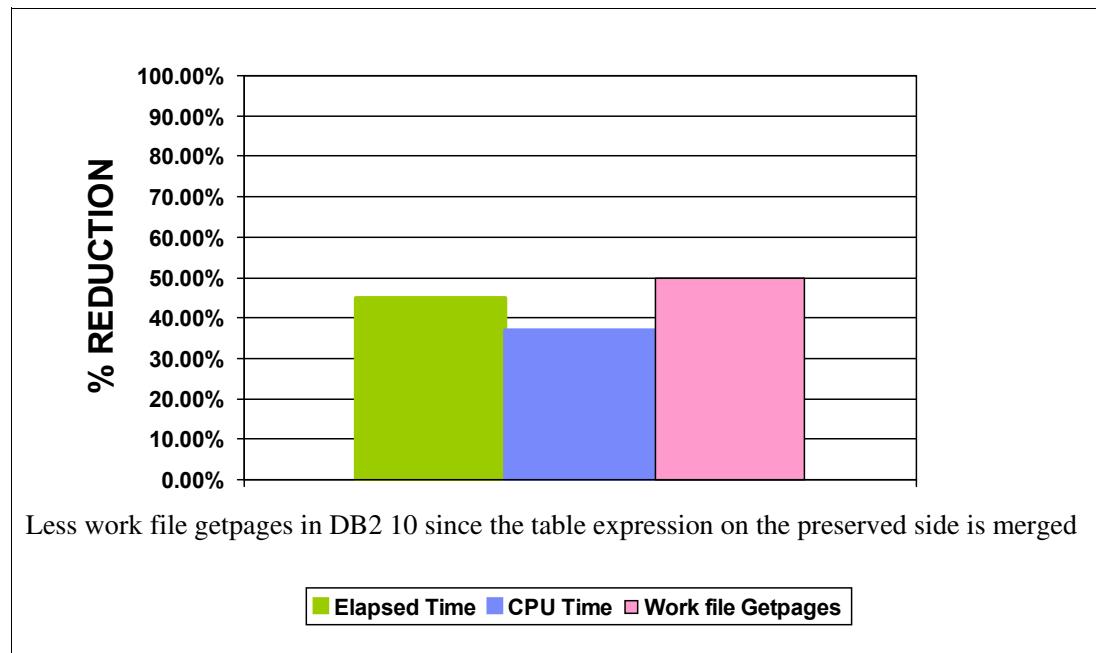


Figure 6-24 Performance for merge of table expression on preserved side of outer join

The aggressive merge of the table expression on the preserved side of an outer join results in a 45% reduction in elapsed time, a 37% reduction in CPU time and a 50% reduction in work file getpages.

DB2 10 can provide significant savings by avoiding materialization of work files for views and table expressions in certain situations. See *DB2 10 for z/OS Technical Overview*, SG24-7892, for details and more examples of when DB2 10 can avoid materialization.

The enhancements to aggressively merge views and table expressions to avoid materialization to work files is available in conversion mode after rebind.

6.8 Implicit casting extension

DB2 10 for z/OS extends the compatibility rule for character or Unicode graphic string and numeric data types. Starting with DB2 10 new-function mode, the character or Unicode graphic strings and numeric data types are compatible, and DB2 performs an implicit cast between the values of those data types.

DB2 performs implicit casting in many cases. For example, when inserting an INTEGER value into a DECIMAL column, the INTEGER value is implicitly cast to the DECIMAL data type. Therefore, you can assign a character or Unicode graphic sting value directly to a target with a numeric data type or compare it with a numeric value. Conversely, you can assign a numeric value directly to a target of a character or Unicode graphic string data type.

When DB2 implicitly casts a numeric value to a string value, the target data type is VARCHAR, which is then compatible with other character or Unicode graphic string data types. The length attribute and the CCSID attribute of the result are determined in the same way as for the VARCHAR built-in scalar function. Note that when implicitly casting a numeric value to a graphic string, the graphic string must be of Unicode encoding.

When DB2 implicitly casts a character or Unicode graphic string to a numeric value, the target data type is DECFLOAT(34), which is then compatible with other numeric data types. The reason DECFLOAT(34) was chosen for the target data type is that a valid string representation of any numeric value (integer, decimal, floating-point or other numeric value) is also a valid representation of a DECFLOAT(34) number, but not the other way around. In other words, the set of all string representations of a DECFLOAT(34) value is a proper superset of the set of all string representations of other numeric data types. Note that when implicitly casting a graphic string to a numeric value, the graphic string must be of Unicode encoding.

Table 6-6 shows the implicit cast target data types and length.

Table 6-6 Implicit cast target data types and length

Source data type	Target data type
SMALLINT	VARCHAR(6)
INTEGER	VARCHAR(11)
BIGINT	VARCHAR(20)
DECIMAL(p,s)	VARCHAR(p+2)
REAL, FLOAT, DOUBLE	VARCHAR(24)
DECFLOAT	VARCHAR(42)
CHAR, VARCHAR, GRAPHIC, VARGRAPHIC	DECFLOAT(34)

Implicit casting is intended as a usability enhancement, not a performance enhancement. There is a price you pay in performance when DB2 cannot directly cast from the source data type to the target data type and has to go through extra processing. You need to be aware of the data types used in your SQL statements and understand how implicit casting works to ensure that your SQL statements are not incurring unnecessary overhead. For details on the processing associated with implicit casting, see the section on implicit casting in the *DB2 10 for z/OS Technical Overview*, SG24-7892.



Application enablement

In this chapter, we describe the performance of DB2 10 for z/OS enhancements that are application specific, but not strictly confined to SQL.

These enhancements provide infrastructure support for new applications or simplify the portability of existing applications to DB2 for z/OS from other database systems.

In this chapter, we discuss the following topics:

- ▶ Temporal support
- ▶ Referential integrity checking improvements
- ▶ Support for `TIMESTAMP WITH TIMEZONE`
- ▶ Additional non-key columns in a unique index
- ▶ Dynamic SQL literal replacement
- ▶ EXPLAIN MODE special register to explain dynamic SQL
- ▶ Access plan stability
- ▶ Access currently committed data

7.1 Temporal support

The concept of associating data with a point in time is paramount to practically every aspect of today's business (that is, business time). With ever growing data with respect to time (that is, system time), there is a strong demand for temporal capability inside the database management system. In DB2 10 for z/OS NFM, the temporal capability introduces the functionality of a table being defined with attributes pertaining to time. This functionality is available to UTS, classic partitioned, and single table segmented table spaces.

Temporal tables with their built-in capability automatically understand the business time and system time that the data was entered into the system. This capability is ideally suited for finding out the condition of the business/data as of a certain time. There are two new BUSINESS_TIME and SYSTEM_TIME table period definition columns available. These new time period column definitions are used for the new DB2 temporal table definitions to provide system-maintained, period-maintained or bitemporal (when both system and period maintained) database tables.

Without this temporal capability provided by DB2, application developers and database administrators must manage different versions of application data. Such tasks can involve complicated, difficult-to-maintain scenarios (for example, many triggers or complex application logic of keeping copies of the tables/data in synch) and might pose performance concerns. Temporal capability can reduce the complexity of applications that do or plan to do this type of processing and can improve application performance by moving the logic from application level into the database system level.

7.1.1 New table attributes

In order to use versioning to keep historical rows, there is a new concept called *period*. A period is implemented using two columns:

- ▶ One column contains the start value of a period.
- ▶ The other column contains the end value of a period.

The allowed data type for those columns depends on the type of the period and is either a DATE, a TIMESTAMP(6) WITHOUT TIMEZONE, or a TIMESTAMP(12) WITHOUT TIMEZONE. Time zone support for periods is not provided at this time; all values in those columns are local date/time values and are not in UTC.

There are two types of time periods that DB2 supports:

System period A pair of columns with system-maintained values that indicate the period of time when a row is valid, which can also be referred to as a SYSTEM_TIME period.

Application period A pair of columns with application-maintained values that indicate the period of time when a row is valid, which can also be referred to as a BUSINESS_TIME period.

Each of the two period types is geared towards supporting a specific temporal capability:

- ▶ The system period is intended for supporting a concept of data versioning and is used for providing the system maintained history of the data.
- ▶ The application period is intended for supporting a user-controlled way of establishing the notion of validly of the data. It allows user applications to specify and control the periods when certain data is considered valid to the user.

Depending on attributes used for the table definition, the following three types of tables are possible:

- ▶ System period temporal tables:

DB2 tables that contain current active rows. The system determines the migration of the updated rows to a history table, sets the start and end time of a period and is based on insert, update, delete events using a built-in period called SYSTEM_TIME

- ▶ Business period temporal tables:

Tables that allow user application to set the paired time values through BUSINESS_TIME period. This might also be referred to as “Application period temporal tables” because the application/users, not the DB2 system, control what values are stored for the business period.

- ▶ Bitemporal tables:

Tables that support both SYSTEM_TIME and BUSINESS_TIME period.

7.1.2 Data versioning

To enable versioning of a system maintained temporal table, the table needs to be created (or altered) as shown in Example 7-1. After versioning has been established, then querying of historical data can be done.

Example 7-1 DDL code to enable versioning of data - System Time temporal capability of DB2

```
--create a base table with system time period
CREATE TABLE policy_detail
(policy_id INTEGER,
amount DEC(12,2),
start_ts TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
end_ts TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
trans_id TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
PERIOD SYSTEM_TIME(start_ts,end_ts));

--create history table with the same columns as the base table, but without the
--system time period
CREATE TABLE hist_policy
(policy_id INTEGER,
amount DEC(12,2),
start_ts TIMESTAMP(12) NOT NULL,
end_ts TIMESTAMP(12) NOT NULL,
trans_id TIMESTAMP(12));

--alter table to pair up the base table and history table
ALTER TABLE policy_detail ADD VERSIONING USE HISTORY TABLE hist_policy;
```

If a current application implements data versioning, it is a good idea to migrate that application to this new system way of versioning as this can potentially improve the overall performance and at the same time simplify the user tasks for maintaining and querying the historical data.

7.1.3 Application performance comparisons

The following test environment on a native z10 system was used in the performance comparisons discussed in this section:

- ▶ 3 regular 2097 z10 CP, 1 zIIP, 1 zAAP
- ▶ 32 GB real storage
- ▶ z/OS 1.11
- ▶ Gigabit Ethernet network connectivity
- ▶ DS8300 DASD

All these tests were carried out on a DB2 10 for z/OS subsystem, even for the cases where user defined triggers were used to simulate the temporal versioning behavior.

DB2 provided system time support versus user defined triggers: Mixed load

In order to compare the performance of DB2 provided system time temporal implementation versus that of user defined solution, the following three test runs were carried out and the results, in terms of CPU and elapsed time (msec) tabulated in Table 7-1. All three tests were carried out with the same workload, which is a transaction mix of 70% read (SELECT), 30% write (10% INSERT + 20% UPDATE/DELETE).

Run#1: Baseline workload without history table

This test was performed to create a baseline for the CPU time and elapsed time for the DML activities on the base tables alone using regular table definitions.

Run#2: User defined trigger solution with history

In this test, user defined trigger solution was used for maintaining history data, that is, temporal feature was not exploited. The history tables were created and the updates to the base table were synchronized using AFTER UPDATE and AFTER DELETE triggers to simulate the temporal versioning behavior. The elapsed time and CPU time are tabulated in Table 7-1.

Run#3: System time temporal support with history

For this run, all the triggers were dropped, and after altering the tables to support SYSTEM TIME, the same workload was run against the temporal tables.

The results from all three of the test runs are listed in Table 7-1.

Table 7-1 System time temporal versus user defined trigger solution on a mixed workload

Test	Scenario	Elapsed time (msec)	CPU time (msec)
Run#1	Baseline workload without history	70.75	49.86
Run#2	User defined trigger Solution with history	112.32	78.31
Run#3	System time support with history	84.23	62.33

Savings in maintaining the history tables using temporal capability

The trigger CPU overhead can be calculated as $78.31 - 49.86 = 28.45$. The CPU overhead associated with temporal history maintenance can be calculated as $62.33 - 49.86 = 12.47$. So, the percentage savings in CPU overhead can be calculated as $(28.45 - 12.47) / 28.45 = 56\%$ on this mixed workload.

Figure 7-1 illustrates the results from these test runs on a bar graph.

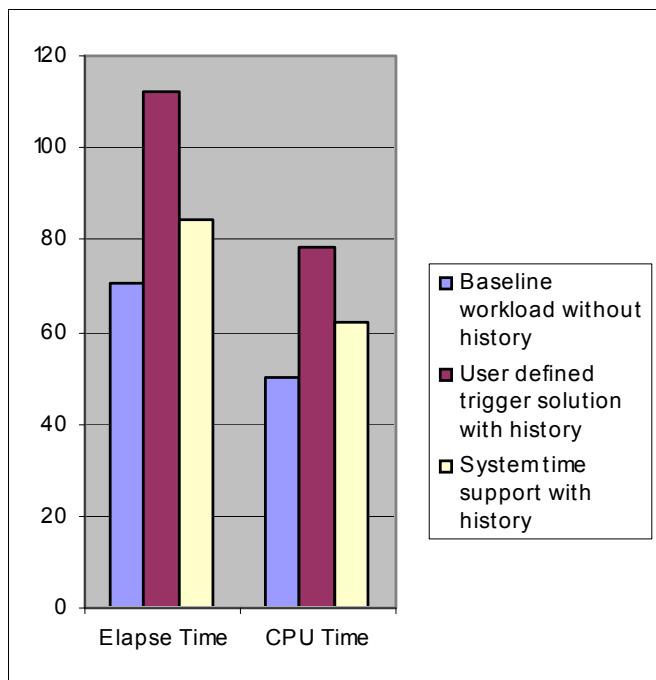


Figure 7-1 DB2 system time temporal versus user defined trigger solution for a mixed workload

UPDATE performance with SYSTEM TIME temporal versus trigger solution

Two tests were run to compare the performance of UPDATE statements with and without temporal capability. The first test is to run the UPDATE statement on a regular table with triggers defined on it to simulate system time temporal behavior using the SQL statements shown in Example 7.1.

Example 7-2 SQL statements used for trigger solution on a regular table UPDATE

```
CREATE TABLE EMP (EMPNO INT NOT NULL,  
                  SALARY DEC(8,2) NOT NULL,  
                  SYS_BEGIN TIMESTAMP(12) NOT NULL,  
                  SYS_END   TIMESTAMP(12) NOT NULL,  
                  TRANS_ID  TIMESTAMP(12))!  
CREATE TABLE EMP_HIST (EMPNO INT NOT NULL,  
                      SALARY DEC(8,2) NOT NULL,  
                      SYS_BEGIN TIMESTAMP(12) NOT NULL,  
                      SYS_END   TIMESTAMP(12) NOT NULL,  
                      TRANS_ID  TIMESTAMP(12))!  
INSERT INTO EMP(EMPNO, SALARY, SYS_BEGIN, SYS_END, TRANS_ID)  
VALUES(1, 40000,  
'2010-02-22-00:00:00.000000000000',  
'9999-12-31-24:00:00.000000000000',  
'2010-02-22-00:00:00.000000000000')!  
  
CREATE TRIGGER TRIG01  
NO CASCADE BEFORE UPDATE ON EMP  
REFERENCING  
      NEW AS Y  
FOR EACH ROW MODE DB2SQL
```

```

BEGIN ATOMIC
    SET Y.SYS_BEGIN = CURRENT TIMESTAMP;
    SET Y.TRANS_ID  = CURRENT TIMESTAMP;
    SET Y.SYS_END   = '9999-12-31-24.00.00.000000';
END!

CREATE TRIGGER TRIG02
AFTER UPDATE ON EMP
REFERENCING OLD AS OLD_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
    INSERT INTO EMP_HIST(EMPNO, SALARY, SYS_BEGIN, SYS_END, TRANS_ID) VALUES
    (OLD_ROW.EMPNO,OLD_ROW.SALARY,OLD_ROW.SYS_BEGIN,CURRENT
TIMESTAMP,OLD_ROW.TRANS_ID);
END !

--actual UPDATE statement used in the test scenario

UPDATE EMP SET SALARY = 88888 WHERE EMPNO=1!

```

The second test is to run the UPDATE statement against the temporal table with data versioning enabled. The UPDATE statement along with the corresponding DDL statements used are shown in Example 7-3.

Example 7-3 SQL statements used for system time temporal table UPDATE

```

CREATE TABLE EMP (EMPNO INT NOT NULL,
                  SALARY DEC(8,2) NOT NULL,
                  SYS_BEGIN TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
                  SYS_END   TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
                  TRANS_ID  TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
                  PERIOD SYSTEM_TIME(SYS_BEGIN, SYS_END));
CREATE TABLE EMP_HIST (EMPNO INT NOT NULL,
                      SALARY DEC(8,2) NOT NULL,
                      SYS_BEGIN TIMESTAMP(12) NOT NULL,
                      SYS_END   TIMESTAMP(12) NOT NULL,
                      TRANS_ID  TIMESTAMP(12));
ALTER TABLE EMP ADD VERSIONING USE HISTORY TABLE EMP_HIST;
INSERT INTO EMP VALUES(1, 40000);

--actual UPDATE statement used in the test scenario
UPDATE EMP SET SALARY = 88888 WHERE EMPNO=1;

```

The two tests were carefully repeated to update 10 rows and then to update 80 rows, and the results were plotted in a bar graph as shown in Figure 7-2. In this figure, and the two following for DELETE and INSERT, we assume 100 the measurement of DB2 class 2 CPU time for the base case and show the difference in % with the other two cases.

The performance results were plotted with the corresponding UPDATE statements against the regular table that has equivalent size and number of indexes as the temporal table.

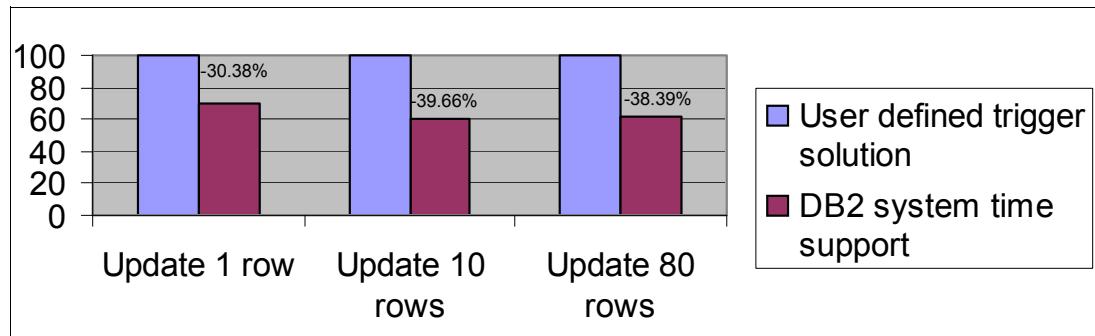


Figure 7-2 UPDATE performance with SYSTEM TIME temporal versus trigger solution

Even though significant UPDATE path length and CPU time overhead are expected with the data versioning (temporal) function enabled, UPDATE with SYSTEM TIME support performed better than using trigger for history row backup.

For the test scenario described in this section, DB2-provided system time temporal support for an UPDATE out-performed user defined trigger solution by 30% to 39%.

DELETE performance with SYSTEM TIME temporal versus trigger solution

Two tests were run to compare the performance of DELETE statements with and without temporal capability. In the first test, the DELETE statement was run on a regular table, which had triggers defined on it to simulate system time temporal behavior using the SQL statements shown in Example 7-4.

The performance results were recorded with the same DELETE statement against the regular table that has equivalent size and number of indexes as that of the temporal table.

Example 7-4 SQL statements used for trigger solution on a regular table DELETE

```

CREATE TABLE EMP (EMPNO INT NOT NULL,
                  SALARY DEC(8,2) NOT NULL,
                  SYS_BEGIN TIMESTAMP(12) NOT NULL,
                  SYS_END   TIMESTAMP(12) NOT NULL,
                  TRANS_ID  TIMESTAMP(12))!
CREATE TABLE EMP_HIST (EMPNO INT NOT NULL,
                      SALARY DEC(8,2) NOT NULL,
                      SYS_BEGIN TIMESTAMP(12) NOT NULL,
                      SYS_END   TIMESTAMP(12) NOT NULL,
                      TRANS_ID  TIMESTAMP(12))!
INSERT INTO EMP(EMPNO, SALARY, SYS_BEGIN, SYS_END, TRANS_ID)
VALUES(1, 40000,
      '2010-02-22-00:00:00.000000000000',
      '9999-12-31-24:00:00.000000000000',
      '2010-02-22-00:00:00.000000000000')!
CREATE TRIGGER TRIG01
AFTER DELETE ON EMP
REFERENCING OLD AS OLD_ROW
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
  INSERT INTO EMP_HIST(EMPNO, SALARY, SYS_BEGIN, SYS_END, TRANS_ID) VALUES
  (OLD_ROW.EMPNO,OLD_ROW.SALARY,OLD_ROW.SYS_BEGIN,CURRENT
  TIMESTAMP,OLD_ROW.TRANS_ID);

```

```

END !

--actual DELETE statement used in the test scenario

DELETE FROM EMP WHERE EMPNO=1!

```

The second test is to run the DELETE statement against the temporal table with data versioning enabled. The actual DELETE statement used along with the corresponding DDL statements are shown in Example 7-5.

Example 7-5 SQL statements used for System time temporal table DELETE

```

CREATE TABLE EMP (EMPNO INT NOT NULL,
                  SALARY DEC(8,2) NOT NULL,
                  SYS_BEGIN TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW BEGIN,
                  SYS_END   TIMESTAMP(12) NOT NULL GENERATED ALWAYS AS ROW END,
                  TRANS_ID  TIMESTAMP(12) GENERATED ALWAYS AS TRANSACTION START ID,
                  PERIOD SYSTEM_TIME(SYS_BEGIN, SYS_END));
CREATE TABLE EMP_HIST (EMPNO INT NOT NULL,
                      SALARY DEC(8,2) NOT NULL,
                      SYS_BEGIN TIMESTAMP(12) NOT NULL,
                      SYS_END   TIMESTAMP(12) NOT NULL,
                      TRANS_ID  TIMESTAMP(12));
ALTER TABLE EMP ADD VERSIONING USE HISTORY TABLE EMP_HIST;
COMMIT;
INSERT INTO EMP(EMPNO, SALARY) VALUES(1, 40000);
COMMIT;

--actual DELETE statement used in the test scenario

DELETE FROM EMP WHERE EMPNO=1;

```

The two tests were carefully repeated, but changed to delete 10 rows and then to delete 80 rows, and the results were plotted in a bar graph as shown in Example 7-3.



Figure 7-3 DELETE performance with SYSTEM TIME temporal versus trigger solution

Even though significant DELETE path length and CPU time overhead are expected with the data versioning (that is, temporal) function enabled, DELETE with SYSTEM TIME support performed better than that of using a trigger solution for history row backup.

For the test scenario described in this section, DB2-provided system time temporal support for DELETE out-performed user defined trigger solution by 10% to 19%.

Update performance with business time temporal versus stored procedure

Two tests were run to measure the UPDATE performance difference on a Business Time Temporal Table as described next.

First we ran the UPDATE statement against a temporal table with a business time period defined. The update operation will cause a two row split. The UPDATE statement used for this test are shown in Example 7-6.

Example 7-6 UPDATE statement on business time temporal table

```
UPDATE temporal_policy
FOR PORTION OF BUSINESS_TIME
FROM DATE('2010-02-13') TO DATE('2010-02-15')
SET PREMIUM = 888 WHERE policy_id = 1234
```

We then ran the equivalent UPDATE statement against a regular table, which is of same size and with same number of indexes as the business time temporal table, but uses a pre-defined stored procedure to simulate the row split behavior. The UPDATE statement, trigger, stored procedure, and the corresponding DDL statements used for this test are shown in Example 7-7.

Example 7-7 SQL for UPDATE performance on a regular table with stored procedure solution

```
CREATE TABLE policy (policy_id INT NOT NULL,
                     PREMIUM DEC(8,2) NOT NULL,
                     BUS_BEGIN DATE NOT NULL,
                     BUS_END   DATE NOT NULL,
                     TEMP_BEGIN DATE,
                     TEMP_END   DATE,
                     PRIMARY KEY (policy_id, BUS_BEGIN, BUS_END));
INSERT INTO EMP(policy_id, premium, BUS_BEGIN, BUS_END) VALUES(1, 30000,
'1999-07-20', '2005-10-13');
INSERT INTO EMP(policy_id, premium, BUS_BEGIN, BUS_END) VALUES(1, 30000,
'2005-10-13', '2010-10-13');

CREATE PROCEDURE update_policy_premium
  (IN update_bus_begin      DATE,
   IN update_bus_end        DATE,
   IN old_bus_begin         DATE,
   IN old_bus_end           DATE,
   IN update_premium        DECIMAL(8,2),
   IN old_premium            DECIMAL(8,2),
   IN old_policy_id          INT)
  LANGUAGE SQL
  MODIFIES SQL DATA
BEGIN

  -- no row split: update duration of existing row
  -- |-----|0000000000000000|---->
  -- |-----|UUUUUUUUUUUUUUUU|---->
  -- DO NOTHING!!!

  -- 3 rows split:
  -- |-----|0000000000000000|---->
  -- |-----|1.00|2.UUUU|3.00|---->
```

```

        IF ((update_bus_begin > old_bus_begin) and (update_bus_end < old_bus_end))
THEN
        INSERT INTO policy (policy_id, premium, bus_begin, bus_end) VALUES
          (old_policy_id, old_premium, old_bus_begin, update_bus_begin);
        INSERT INTO policy (policy_id, premium, bus_begin, bus_end) VALUES
          (old_policy_id, old_premium, update_bus_end, old_bus_end);
      END IF;

      -- 2 row split, update of the last half portion
      -- |-----|0000000000000000|---->
      -- |-----|00000000|UUUUUUU|---->

      IF ((update_bus_begin > old_bus_begin) and (update_bus_end >= old_bus_end))
THEN
        INSERT INTO policy (policy_id, premium, bus_begin, bus_end) VALUES
          (old_policy_id, old_premium, old_bus_begin, update_bus_begin);
      END IF;

      -- 2 row split, update of the first half portion
      -- |-----|0000000000000000|---->
      -- |-----|UUUUUUUU|0000000|---->

      IF ((update_bus_begin <= old_bus_begin) and (update_bus_end < old_bus_end))
THEN
        INSERT INTO policy (policy_id, premium, bus_begin, bus_end) VALUES
          (old_policy_id, old_premium, update_bus_end, old_bus_end);
      END IF;
END
@

CREATE TRIGGER update_business_period
AFTER UPDATE ON EMP
REFERENCING NEW AS N
OLD AS O
FOR EACH ROW
MODE DB2SQL
BEGIN ATOMIC
  CALL update_policy_premium(N.TEMP_BEGIN, N.TEMP_END, O.BUS_BEGIN, O.BUS_END,
N.premium, O.premium, O.policy_id);
END
@
--actual UPDATE statement used in the test scenario

UPDATE policy SET PREMIUM = 50000,
TEMP_BEGIN = '1999-07-21',
TEMP_END   = '2010-10-13'
WHERE policy_id = 1
AND(DATE('1999-07-21') >= BUS_BEGIN AND DATE('1999-07-21') < BUS_END)
OR (DATE('2010-10-13') > BUS_BEGIN AND DATE('2010-10-13') <= BUS_END)
OR (DATE('1999-07-21') <= BUS_BEGIN AND DATE('2010-10-13') >= BUS_END);

```

The two tests were carefully repeated but, this time the update was coded to cause a 3-row split. The results from both sets of tests were plotted in a bar graph as shown in Figure 7-4.

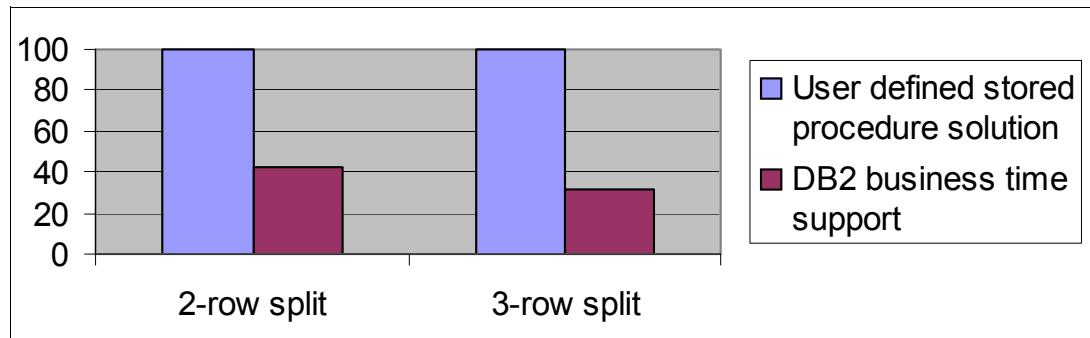


Figure 7-4 UPDATE performance on business time temporal versus stored procedure solution

The UPDATE path length and CPU time with BUSINESS TIME support was observed to be better than that of the equivalent stored procedure solution.

INSERT WITH and WITHOUT OVERLAPS on business time temporal

This section shows the performance difference observed on INSERTs into a business time temporal table with a “BUSINESS_TIME WITHOUT OVERLAPS” index when compared to a conventional index, which allows overlaps. See Figure 7-5.

If BUSINESS_TIME WITHOUT OVERLAPS is specified, the values for the rest of the specified keys are unique with respect to the time for the BUSINESS_TIME period.

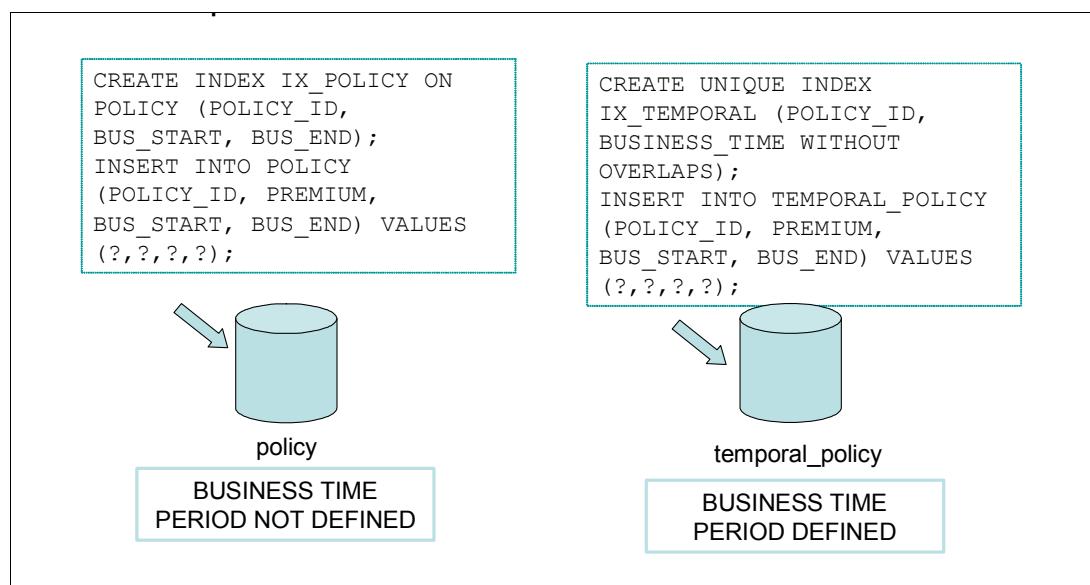


Figure 7-5 INSERT performance WITH versus WITHOUT OVERLAPS index

Less than 1% of CPU degradation was observed. Thus, in order to ensure uniqueness of the business time period, DB2 incurs approximately 1% overhead on INSERT statements.

Note that referential constraints can be specified on temporal tables with a BUSINESS_TIME period, but there is no support for having a referential constraint relationship across periods of time (that is, no foreign key specification of BUSINESS_TIME WITHOUT OVERLAPS). The enforcement of uniqueness over a period of time is the important functionality delivered with temporal tables with BUSINESS_TIME periods. It is optional for a user to have unique BUSINESS_TIME WITHOUT OVERLAPS. If none is specified, then overlaps in time can occur and DB2 does not do any overlap checking.

System time temporal sample queries and access path

Table 7-2 shows three sample queries which return identical rows, along with their respective run time. We compare the execution of RDS transformed query with base table UNION ALL with history table as a result of SYSTEM_TIME AS OF syntax versus user specified query with explicit base table UNION ALL with history table.

At the time of running the queries in Table 7-2, the following conditions existed:

- ▶ LINEITEM table had 6 million rows while LINEITEM_H had 18 million.
- ▶ Both tables were defined with a clustering index on L_ORDERKEY column.
- ▶ The buffer pool is cleaned up before each query to prevent skewed results.

Table 7-2 Temporal queries with explicit and implicit UNION ALL with history tables

No	Query	UNION ALL	DB2 elapsed time	DB2 CPU time
1	SELECT * FROM LINEITEM WHERE L_ORDERKEY=80001 AND SYS_BEGIN < '2009-10-06-02.00.58' UNION ALL SELECT * FROM LINEITEM_H WHERE L_ORDERKEY=80001 AND SYS_END < '2009-10-06-02.00.58'	Explicit	1.382320	0.009238
2	SELECT * FROM LINEITEM FOR SYSTEM_TIME AS OF '2009-10-06-02.00.58' WHERE L_ORDERKEY = 80001;	Implicit - due to AS OF predicate	1.372669	0.009321
3	SELECT * FROM LINEITEM FOR SYSTEM_TIME BETWEEN '2001-01-01-02.00.58' AND '2009-10-06-02.00.58' WHERE L_ORDERKEY = 80001;	Implicit - due to date range predicate	1.608537	0.009156

The performance difference is insignificant for the three types of queries tabulated in Table 7-2. The general performance expectations of System Time Temporal SELECT statements can be summarized as follows:

- ▶ SELECT queries against base tables (that is, current data) will perform very similar to tables not performing data versioning.
- ▶ SELECT queries accessing historical data using the temporal syntax might be slower (in terms of DB2 CPU time). The performance overhead of temporal SQL statement is determined by efficiency of indexing on both base and history table, type of join, and number of table being joined.

The access path for all the three SQL statements listed in Table 7-2 are very similar, as can be seen in Figure 7-6, Figure 7-7, and Figure 7-8. The implicit UNION ALL can be explicitly seen in the access path information.

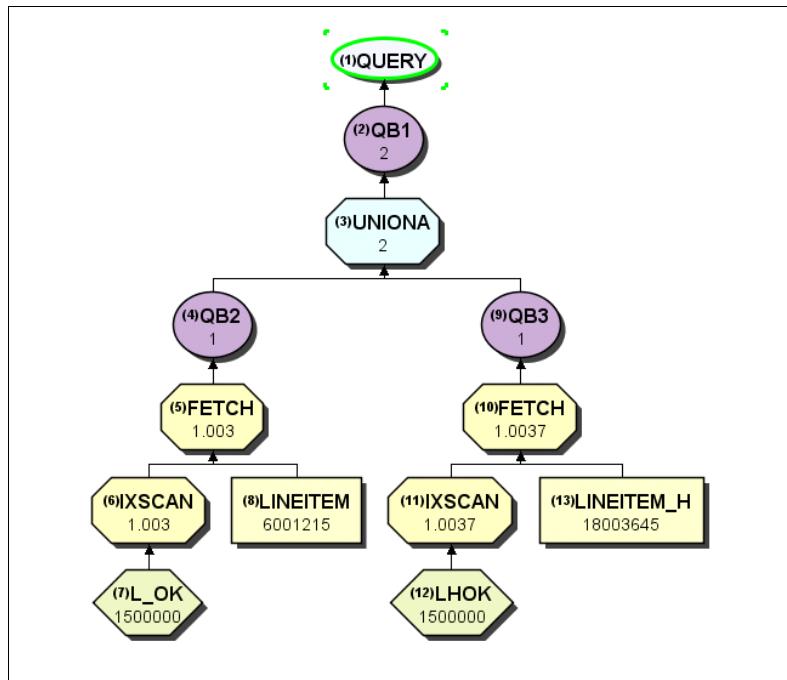


Figure 7-6 System Time Temporal SELECT statement#1 - Access path

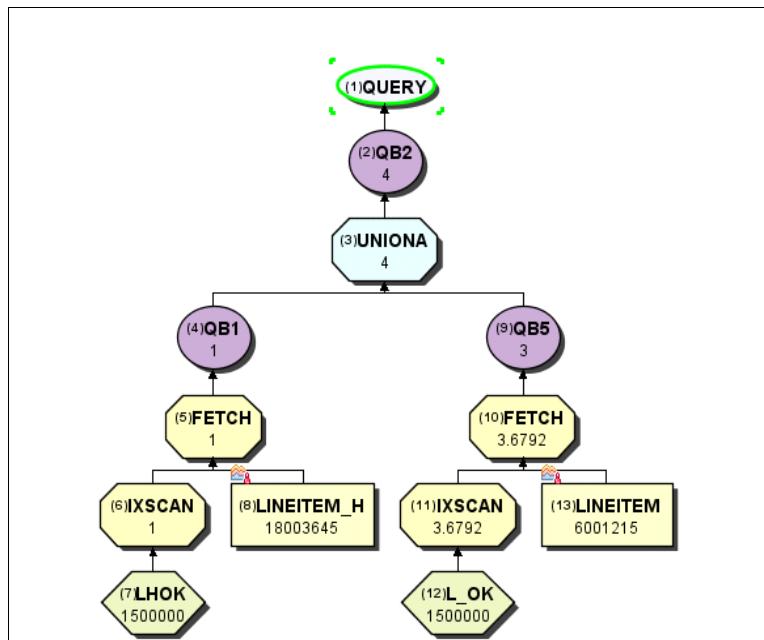


Figure 7-7 System Time Temporal SELECT statement#2 - Access path

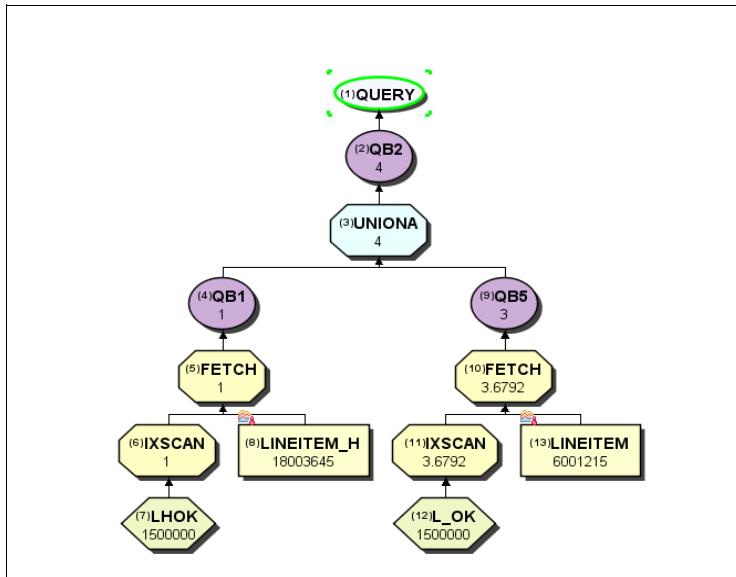


Figure 7-8 System Time Temporal SELECT statement#3 - Access path

The sample statement#4 is shown in Example 7-8.

Example 7-8 Sample SQL accessing base (temporal) table - SQL statement#4

```
SELECT *
FROM USRT002.LINEITEM
WHERE L_ORDERKEY = 80001;
```

Figure 7-9 shows a sample access path diagram for the sample query on LINEITEM (temporal) table (in Example 7-8). This query results in accessing the base table only as shown in the diagram and is provided for the sake of comparison with the other three access path diagrams.

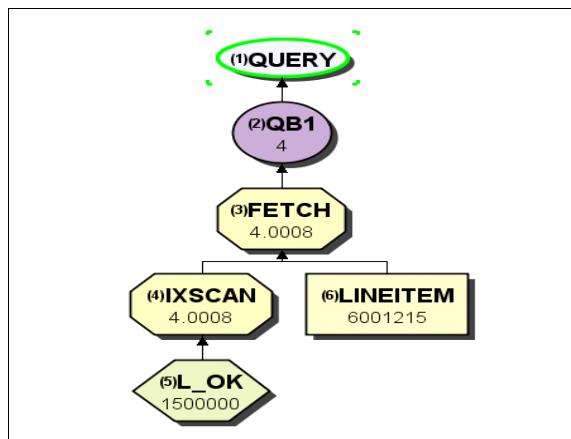


Figure 7-9 System Time Temporal SELECT statement#4 - Access path (base table only)

7.1.4 Productivity improvements with temporal feature

Productivity improvements for application programmers are very important because people are the largest component of cost when it comes to application development.

The new bitemporal capability for data provides the ability for a table to contain both current and historical data, and to query the information AS OF a specific point in time. This feature naturally provides historical auditing capabilities and drastically improves the productivity of valuable resources.

In general, DB2 business time and system time period support will greatly reduce cost and labor required for application development. For a sample scenario studied, it was observed that both business time and system time period support outperform equivalent functions implemented with user defined triggers or stored procedures.

Table 7-3 summarizes the productivity savings that can be expected while using the Temporal capabilities of DB2 rather than a typical user application solution.

Table 7-3 Productivity improvements using temporal capability

Comparison	User application solution	DB2 supplied solution
System time support	2 triggers for each table	Versioning capability
Business time support	2 stored procedures for each table	SQL statements
Period overlap detection	1 trigger for each table	Part of primary key index for each table
Total number of lines of code	650	13
Total number of SQL statements	27	3
Time to develop and test	7 weeks	<1 hour

7.1.5 Improved data warehousing capabilities

DB2 offers application designers new functionality for their data warehousing requirements through the bitemporal capability.

The temporal feature for SQL provides easier expression of time based queries and also more efficient execution of such queries by DB2 engine rather than the application, thus making data warehousing more efficient.

7.1.6 Summary on temporal support

DB2 provides a capability to specify table-level specifications to control the management of application data based upon time.

Application programmers can specify a search criteria based upon the time the data existed or was valid. This function simplifies DB2 application development requiring data versioning.

Customers can satisfy compliance laws faster and cheaper because DB2 automatically manages the different versions of data.

The support for system period temporal tables and business period temporal tables is available in new function mode.

The performance expectations along with overhead associated with both System and Business Period temporal table usage are summarized next (your mileage might vary depending on which method is being used as the baseline to compare).

System period temporal tables

The performance expectations are as follows:

- ▶ DB2-provided system time support for UPDATE out-performs user defined triggers by 30 – 39%, and by 10-19% for DELETE respectively in DB2 CPU time.
- ▶ UPDATE and DELETE against base table performance is affected by the history insert performance, which in turn is controlled by the number of indexes, number of columns in the index, clustering order, space map search, and other information on the history table.
- ▶ INSERT and UPDATE of current data (that is, from the base table) might perform slower than tables not performing data versioning.
- ▶ SELECT against base tables (that is, current data) will perform similar to tables not performing data versioning.
- ▶ SELECT of historical data using the new approach might be slower. The performance overhead of temporal query statement is determined by efficiency of indexing on both base and history table, type of join, and number of table joining.

Business period temporal tables

The performance expectations are as follows:

- ▶ 1-3% overhead of maintaining business periods without overlaps can be expected.
- ▶ DB2-provided business time support for row splitting out-performs user defined stored procedure by 57% to 68% in DB2 CPU time.

Also, the following topics pertaining to temporal feature have been discussed in this section:

- ▶ Application performance comparisons
- ▶ Productivity improvements with temporal feature
- ▶ Improved data warehousing capabilities
- ▶ Summary on temporal support

7.2 Referential integrity checking improvements

When inserting into a dependent table, DB2 must access the parent key for referential constraint checking. DB2 10 uses dynamic prefetch and reduces the get pages on indexes and related CPU overhead by minimizing index probes for parent keys.

The enhancements are as follows:

- ▶ DB2 10 allows sequential detection to trigger dynamic prefetch for parent key referential integrity checking. This enhancement potentially improves elapsed time by detecting the need for prefetch and enabling it when necessary.

- ▶ When referential integrity is defined for a parent and child table, and new rows are inserted into the child table, DB2 has to validate the values in the foreign key columns.

Prior to DB2 10, this referential integrity checking is always performed against an index on the parent table. During this checking, index look-aside is not enabled even when multiple inserts are performed in key sequence.

DB2 10 enables index look-aside for parent key referential integrity checking. Index look-aside is when DB2 caches key range values. DB2 keeps track of the index value ranges and checks whether the required entry is in the leaf page accessed by the previous call. If the entry is found, DB2 can avoid the getpage and traversal of the index tree. If the entry is not found, DB2 checks the parent non-leaf page's lowest and highest key. If the entry is found in the parent non-leaf range, DB2 must perform a getpage but can avoid a full traversal of the index tree.

- ▶ DB2 can also avoid index lookup for referential integrity checking, if the non-unique key to be checked has been checked before.
 - INSERT KEY A, INSERT KEY A,..., INSERT KEY A, COMMIT; For the 1st INSERT KEY A, DB2 checks the parent table index for referential integrity. no referential integrity checking takes place for all subsequent inserts.
 - INSERT KEY A, COMMIT; INSERT KEY A, COMMIT; only the 1st INSERT checks the parent index, all subsequent INSERT will not check.

So for INSERT within or without the same commit scope, if the key is already in the child table, DB2 does not check the parent key value again. If key A is already in the child table (already committed), when you insert another key A (assuming non-unique index on child), DB2 detects that key A is already there, so there is no need to check the parent again because key A already matches the referential integrity rule otherwise it cannot be in the child table.

However, there must be an index on the child table, with the relationship primary key columns defined as leading columns in the index. Otherwise, you will just benefit from index look-aside on the parent table, but not due to the key already existing.

Referential integrity checking can also take advantage of other index enhancements. Hash access can be used for parent key checking. Referential integrity checking is not externalized in the Explain tables.

Two tests were run to measure the performance related to the index probing improvements. In each test, there was a single parent table and a single child table. The buffer pool setup for both tests was as follows:

- ▶ BP11: parent table, tablespace buffer pool
- ▶ BP12: parent table, index buffer pool
- ▶ BP13: dependent table, table space buffer pool
- ▶ BP14: dependent table, index buffer pool

7.2.1 Avoiding checking for existing foreign key values

The scenario for the first test is as follows:

- ▶ The child table contains 5,000 dependent rows that refer to the parent table.
- ▶ The foreign key columns are non-unique.
- ▶ An additional 5,000 rows are inserted into the child table.
- ▶ The 5,000 new rows are inserted by a single INSERT FROM SELECT statement.
- ▶ All 5,000 additional rows have foreign key values that already exist in the child table.

The test showed significant savings as indicated by a reduction in the parent table index getpages from 7,403 to zero. The statistics for the parent table index buffer pool BP12 are shown in Figure 7-10.

DB2 VERSION: SQL DML	DB2 9 TOTAL	DB2 10 TOTAL
SELECT	0	0
INSERT	1	1
ROWS	5000	5000
BP12 BPOOL ACTIVITY	TOTAL	
BPOOL HIT RATIO (%)	100	
GETPAGES	7403	0 getpages
BUFFER UPDATES	0	
SYNCHRONOUS WRITE	0	
SYNCHRONOUS READ	0	
SEQ. PREFETCH REQS	0	
LIST PREFETCH REQS	0	
DYN. PREFETCH REQS	17	
PAGES READ ASYNCHR.	0	

Figure 7-10 Index getpage reduction from avoiding referential integrity checking

The reduction in index getpages is due to DB2 being able to avoid doing referential integrity checking because the foreign key values already exist in the child table.

7.2.2 Exploiting index look-aside

The scenario for the second test is as follows:

- ▶ The child table is empty at the start of the test.
- ▶ The foreign key columns are non-unique.
- ▶ 5,000 rows are inserted into the child table.
- ▶ The 5,000 new rows are inserted by a single INSERT FROM SELECT statement.
- ▶ None of the 5,000 rows have foreign key values that already exist in the child table (because it is empty).

The test showed significant savings as indicated by a reduction in the parent table index getpages from 7,403 to 1,508. The statistics for the parent table index buffer pool BP12 are shown in Figure 7-11.

DB2 VERSION: SQL DML	DB2 9 TOTAL	DB2 10 TOTAL
SELECT	0	0
INSERT	1	1
ROWS	5000	5000
BP12 BPOOL ACTIVITY	TOTAL	TOTAL
BPOOL HIT RATIO (%)	100	100
GETPAGES	7403	1508
BUFFER UPDATES	0	0
SYNCHRONOUS WRITE	0	0
SYNCHRONOUS READ	0	0
SEQ. PREFETCH REQS	0	0
LIST PREFETCH REQS	0	0
DYN. PREFETCH REQS	17	18
PAGES READ ASYNCHR.	0	0

Figure 7-11 Index getpage reduction from index look-aside

The reduction in index getpages is due to DB2 being able to take advantage of index look-aside processing. Index look-aside processing is done when DB2 caches key range values. DB2 keeps track of the index value ranges and checks whether the required entry is in the leaf page accessed by the previous call. If the entry is found, DB2 can avoid the getpage and traversal of the index tree. If the entry is not found, DB2 checks the parent non-leaf page's lowest and highest key. If the entry is found in the parent non-leaf range, DB2 must perform a getpage but can avoid a full traversal of the index tree. The result is that you have less getpages than if index look-aside was not used.

7.2.3 Batch insert with referential integrity

Additional tests were run to measure the CPU reduction due to referential integrity checking improvements. These tests used the same DDL as in the prior tests, but the number of rows inserted were increased to 150,000 to show a more representative sample. The same INSERT FROM SELECT statement was executed, using a replica of the child table as the source table for the inserts:

```
INSERT INTO T45597.CUSTOMER_ACCOUNT    SELECT * FROM T45599.CUSTOMER_ACCOUNT;
```

Child table CUSTOMER_ACCOUNT has a non-unique index on the foreign key. With DB2 9, the first insert of 150,000 rows against the empty child table results in 150,680 getpages against a unique index on the parent table. With DB2 10, the same insert results in 30,282 getpages against a unique index on the parent table. A subsequent insert of 150,000 rows using the same data shows the same 150,680 getpages in DB2 9, while there are no getpages observed in DB2 10 for the index on the parent table for this case.

The first INSERT demonstrates the improvement provided by index look-aside processing as no values have been inserted yet in the child table. We observed a 24% CPU reduction in DB2 10 in this test case compared to DB2 9 by reducing the number of index getpages on the parent table index to about one fifth of what is required for DB2 9. The second INSERT test using the same values demonstrates the improvement from referential integrity lookup avoidance. We see a 28% CPU reduction by completely avoiding the lookup against the parent table index.

The environment used for this test is as follows:

- ▶ z10 LPAR with 5 dedicated CPs
- ▶ Single threaded batch jobs
- ▶ Most of the pages are hit in the buffer pools
- ▶ z/OS R12

The comparison of DB2 elapsed times and DB2 CPU times for the DB2 9 and DB2 10 test cases for referential integrity checking is shown in Figure 7-12.

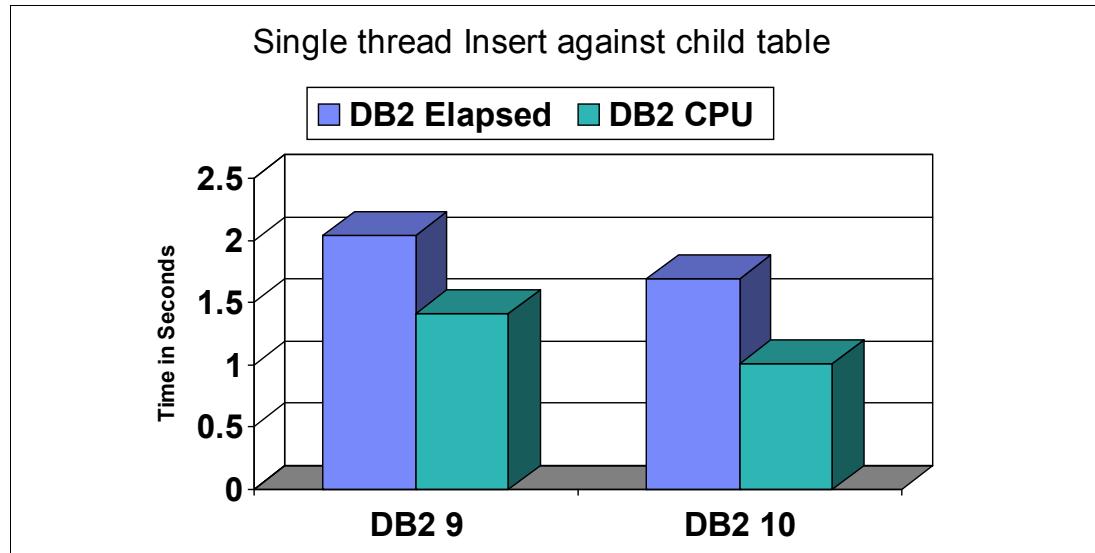


Figure 7-12 DB2 elapsed and CPU time comparison for referential integrity checking enhancements

Note that this test is one of the best scenarios for the referential integrity checking enhancements because there are multiple inserts done against the child table in sequential key order. If the child row inserts are done as one row insert per unit of work, then the improvement from index look-aside and avoidance of referential integrity look up might not be as significant as demonstrated here.

7.2.4 Summary on referential integrity

DB2 10 provides some significant performance enhancements in referential integrity checking, both by avoiding the referential integrity check on the parent altogether if the foreign key already exists in the child and by taking advantage of index look-aside processing to see whether the required index entry is in the leaf page accessed by the previous call. These enhancements can result in a significant reduction in getpages on the parent table index and a significant reduction in both elapsed time and DB2 time.

The referential integrity checking improvements are available in conversion mode.

7.3 Support for TIMESTAMP WITH TIMEZONE

Prior to DB2 10, date and time data types do not include information about the time zone. Customers who need to store date and time data with knowledge of the time zone have to maintain a separate column with the time zone offset.

DB2 10 introduces the new data type TIMESTAMP WITH TIMEZONE. The existing TIMESTAMP data type can be specified as either TIMESTAMP WITHOUT TIMEZONE or just as TIMESTAMP. These two alternatives are identical. DB2 defaults to TIMESTAMP WITHOUT TIMEZONE if just TIMESTAMP is specified.

In addition, DB2 10 provides an enhancement to the TIMESTAMP data type to allow greater precision for timestamp data. You can specify a timestamp precision between 0 digits and 12 digits. The default timestamp precision is 6 digits.

The DDL in Example 7-9 shows how to create a table with a timestamp column that includes time zone.

Example 7-9 Table with time zone data

```
CREATE TABLE TS_TEST_TABLE(
  ID_COL  INTEGER
 ,TMS_9    TIMESTAMP(9)          -- precision 9 without time zone
 ,TMSTZ_9  TIMESTAMP(9) WITH TIMEZONE -- precision 9 with time zone
);
```

Note that both the TIMESTAMP column and the TIMESTAMP WITH TIMEZONE column specify a precision of 9, which is greater than the precision of 6 that is the only option available in DB2 9 and prior versions.

In order to test the performance impact of storing timestamp data with a time zone, we ran a number of tests on DB2 10 against two partition by range (PBR) tables, each with two columns: an INTEGER column as above and either a TIMESTAMP column or a TIMESTAMP WITH TIMEZONE column. Tests were run with various combinations of the following criteria:

- ▶ INSERT or SELECT
- ▶ With or without an index on the appropriate TIMESTAMP column:
- ▶ Dynamic SQL with parameter markers, static with host variable or static with constant

The performance measurements for the queries are shown in Table 7-4. The CPU increase represents the additional cost to use the TIMESTAMP WITH TIMEZONE data type instead of the TIMESTAMP data type.

Table 7-4 Performance measurements for TIMESTAMP WITH TIMEZONE

INSERT/SELECT	Index?	Predicate type	#Rows	%CPU Increase
INSERT	No	Dynamic	1,000,000	5
		Static - Host variable	10,000,000	14
		Static - Constant	1,000,000	5
	Yes	Dynamic	1,000,000	4
		Static - Host variable	10,000,000	9
		Static - Constant	1,000,000	1
SELECT	No	Table space scan	1,000,000	4

In each test there was an additional CPU cost to use the TIMEZONE with the timestamp. The overhead is attributable to the cost of converting between Coordinated Universal Time (UTC) time and TIMESTAMP WITH TIMEZONE value in the bind in or bind out processing. The cost for using a constant versus a host variable is much less because DB2 10 provides a special optimization to convert the constant TIMESTAMP WITH TIMEZONE value to UTC time at bind time to cut down on the cost of conversion at run time.

The new TIMESTAMP WITH TIMEZONE data type is primarily a functional enhancement. However, because applications that need to be aware of time zones no longer need to maintain a separate column for time zone offset and no longer need to adjust times using that offset, the performance cost of using the new data type can be offset by the performance gains of no longer executing the application logic to maintain the time zone. If your applications can benefit from this new data type and can tolerate the cost, then it might be well worth investigating.

The TIMESTAMP WITH TIMEZONE data type is available in new function mode.

7.4 Additional non-key columns in a unique index

Prior to DB2 10, whenever a unique index is created to enforce a uniqueness constraint, all the columns of the index are used to enforce the uniqueness constraint. If you want to use index-only access to improve query performance, and you need additional columns that are not in the unique index, then you need to create a second index including all the columns of the unique index plus the additional columns needed for query performance. The extra index provides improved query performance, but degrades insert and delete performance.

DB2 provides a new capability to INCLUDE non-key columns in a unique index. The additional columns do not participate in the uniqueness constraint, but can be used for index access, including index-only access for certain queries. The intent is to reduce the cost of insert and delete processing by maintaining more indexes. The trade-off is that some queries might result in degraded performance if the additional columns are not used.

To measure the performance benefit of this enhancement on an INSERT workload, we ran a number of INSERT statements on three tables, once using a total of eight unique indexes on the tables and then using a total of four unique indexes with INCLUDED columns. The indexes that were defined for each test case are shown in Figure 7-13.

```

Base - 8 unique indexes

ORDERS
    ORDER_ID, MACHINE_ID
    ORDER_ID, MACHINE_ID, EVENT_ID, ENG_RECEIVE_TIME, ORDER_STATUS
    EVENT_ID, MACHINE_ID
    EVENT_ID, MACHINE_ID, ENG_RECEIVED_TIME

MSG_PROCESSING_SEQ
    MSG_SEQ
    MSG_SEQ, DB_SEND_TIME, MACHINE_ID

ORDER_HISTORY
    EVENT_ID, MACHINE_ID
    EVENT_ID, MACHINE_ID, ORDER_SIDE, FIRM_ID, ENG_RECEIVED_TIME

INCLUDE - 4 unique indexes

ORDERS
    (ORDER_ID, MACHINE_ID) INCLUDE (EVENT_ID, ENG_RECEIVE_TIME, ORDER_STATUS)
    (EVENT_ID, MACHINE_ID) INCLUDE (ENG_RECEIVED_TIME)

MSG_PROCESSING_SEQ
    MSG_SEQ INCLUDE (DB_SEND_TIME, MACHINE_ID)

ORDER_HISTORY
    (EVENT_ID, MACHINE_ID) INCLUDE (ORDER_SIDE, FIRM_ID, ENG_RECEIVED_TIME)

```

Figure 7-13 Index definitions for additional non-key index columns tests

You can see that the number of indexes were reduced from eight for the Base test case to four for the INCLUDE test case. The performance measurements for an INSERT workload comparing the two test cases are shown in Figure 7-14.

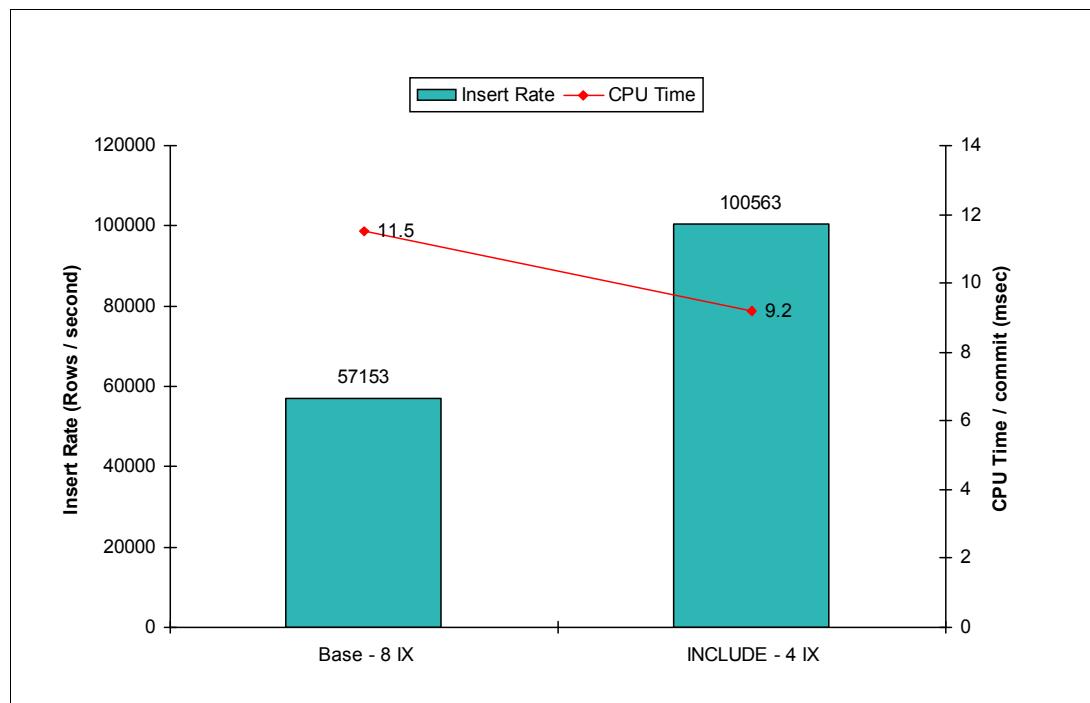


Figure 7-14 INSERT performance measurements for additional non-key index columns

The measurements show that the insert rate (number of rows inserted per second) increased by 75%, while the CPU time for those inserts decreased by 20%.

We cannot measure the benefits of additional non-key columns in a unique index just on the savings of INSERT alone. We also need to look at the impact on query performance.

Consider a query that runs against the ORDER table. In DB2 9 there were two unique indexes defined on the table, as shown in Example 7-10.

Example 7-10 Multiple unique indexes in DB2 9 for additional non-key column query test

--Prior to DB2 10:

```
CREATE UNIQUE INDEX PXO@OK
  ON ORDER (
  O_ORDERKEY )
  USING VCAT TPCD851
  FREEPAGE 0
  PCTFREE 0
  BUFFERPOOL BP12
  CLOSE NO
  CLUSTER
  PARTITIONED ;

CREATE UNIQUE INDEX PXO@OKODCKSPOP
  ON ORDER (
  O_ORDERKEY,O_ORDERDATE,O_CUSTKEY,O_SHIPPRIORITY,O_ORDERPRIORITY )
  USING VCAT TPCD851
  FREEPAGE 0
  PCTFREE 0
  BUFFERPOOL BP12
  CLOSE NO
  PARTITIONED ;
```

In DB2 10 we are able to replace the two unique indexes with a single unique index with additional non-key columns included. The DDL to make those changes is shown in Example 7-11.

Example 7-11 Single unique index in DB2 10 with additional non-key columns

```
DROP INDEX PXO@OK;
DROP INDEX PXO@OKODCKSPOP;

CREATE UNIQUE INDEX PXO@OKINCLUDE ON ORDER
(O_ORDERKEY)
INCLUDE
(O_ORDERDATE,O_CUSTKEY,O_SHIPPRIORITY,O_ORDERPRIORITY)
  USING VCAT TPCD851
  FREEPAGE 0
  PCTFREE 0
  BUFFERPOOL BP12
  CLOSE NO
  CLUSTER
  PARTITIONED ;
```

To test the performance impact of using the INCLUDED columns (PXO@OKINCLUDE) versus using the unique one column index (PXO@OK) for queries that only need the first column, we ran the query in Example 7-12.

Example 7-12 Query to test impact of additional non-key columns in unique index

```
SELECT O_ORDERPRIORITY, COUNT(*)
   FROM ORDER
  WHERE O_TOTALPRICE <= 17500
    AND O_ORDERSTATUS = 'O'
    AND O_ORDERKEY IN (SELECT
                           DISTINCT L_ORDERKEY
                          FROM LINEITEM
                         WHERE L_PARTKEY BETWEEN 10000 AND 12000
                           AND L_LINESTATUS = 'O'
                           AND L_RETURNFLAG = 'N'
                           AND L_COMMITDATE < L_RECEIPTDATE)
  GROUP BY O_ORDERPRIORITY
 ORDER BY O_ORDERPRIORITY;
```

We ran this query twice in DB2 10; once with two separate indexes; once with a single unique index with the additional non-key columns included. The performance measurements for the two runs of the query are shown in Table 7-5.

Table 7-5 Query performance impact for additional non-key columns in unique index

Index used	Elapsed time (sec)	Class 2 CPU time (sec)	DBM1 SRB time (sec)	Getpage requests
PXO@OK	3.60	0.349	1.819	36,874
PXO@OKINCLUDE	4.15	0.658	2.055	39,757
Derived % overhead of INCLUDED columns	15	89	13	8

The performance measurements for INSERT and for SELECT show that there is a trade-off between a good insert rate and good query performance.

This enhancement can improve insert performance because the total number of indexes defined on a table can be reduced with an INCLUDE index. How much improvement you see depends on the size of the index and the number of indexes defined on the table.

You need to evaluate whether query performance is affected. Queries that used to pick the index without the appended columns now use the INCLUDE index. As a result, these queries are impacted by:

- ▶ A larger index being accessed
- ▶ More getpage requests
- ▶ More sync I/O requests
- ▶ Increased CPU cost
- ▶ Increased elapsed time

Following are some considerations for when to include additional columns in a unique index:

- ▶ An index-only scan that uses INCLUDE columns performs the same as an index-only scan that uses ordinary index keys.
- ▶ If an index exists merely to enforce uniqueness, but it is not used by queries, then there is no risk adding some INCLUDE columns to it, because queries are unaffected. But if an index is chosen as the access path, then adding more columns to that index runs the risk of hurting query performance, whether or not the new columns are part of the key.

- ▶ Choosing which indexes to define, and which columns to include in an index, involves a trade-off between insert performance and query performance. Including non-key columns in a unique index does not change this fact.
- ▶ Queries perform better with slim indexes than with fat indexes. A fat index is one that contains extraneous columns that a query does not use. Fat indexes are especially bad for hybrid joins because DB2 does index probes with skip sequential access to the leaf pages. Consequently, the extraneous columns might cause dynamic prefetch to break down, resulting in less prefetch I/O and a lot more synchronous I/O.
- ▶ Using fat indexes is less of a problem for queries that access a single table. A fatter index causes an index scan to do more getpages, but it does not cause a query to do a lot more synchronous I/Os. A fatter index can also cause a worse buffer pool hit ratio for random selects, but inserts might perform better if a fatter index enables you to drop an index.

Allowing additional non-key columns in a unique index is available in new-function mode.

7.5 Dynamic SQL literal replacement

With DB2 9, to take advantage of the dynamic statement cache, the SQL statement string has to be identical and the SQL statement has to be executed by the same user. This enforces the requirement of coding guidelines promoting the use of parameter markers (“?”) in the SQL statement in order to get the maximum performance benefits.

Many applications and development environments generate SQL statements for the developers and often users have no or little control on the way the statements are written. This makes difficult, if not impossible, to exploit parameter markers and such applications will use literals instead. Literals are likely to be different at each SQL statement execution with little reuse of the statement cache. This produces a degradation of performance by requiring a PREPARE at most SQL statement invocation.

In DB2 10 NFM, more SQL statements can be reused in the cache across users. Dynamic SQL statements can now be shared with an already cached dynamic SQL statement if the only difference between the two statements is literal values; when stored in the cache, literals are replaced with an ampersand (&) that behaves similar to parameter markers. This avoids a full PREPARE and can provide a performance improvement similar to what is gained by coding SQL statements with parameter markers.

The normal matching criteria for cached dynamic SQL statements must be the same; for example, the statement must have the same statement length, authorization and BIND options, among other characteristics.

To enable this function, use one of the following methods:

- ▶ On the client, change the PREPARE statement to include the new ATTRIBUTES clause, specifying CONCENTRATE STATEMENTS WITH LITERALS (the acronym CSWL appears in the instrumentation records).
- ▶ On the client side, change the JCC Driver to include the “enableliteralReplacement=“YES”” keyword, which is specified in the data source or connection property.
- ▶ Set LITERALREPLACEMENT in the ODBC initialization file in z/OS, which enables all SQL statements coming into DB2 through ODBC to have literal replacement enabled.

After you have enabled this feature using one of the foregoing methods, the lookup sequence in the dynamic statement cache is as follows:

- ▶ The original SQL statement with literals is looked up in the cache, which is pre-DB2 10 behavior.
- ▶ If not found, the literals are replaced and the new SQL statement is looked up in the cache. DB2 can match only with an SQL statement that is stored with the same attribute. So, any SQL statement in the cache with parameter markers is not matched.
- ▶ If not found, then the new SQL statement is prepared and stored in the cache.

For example, the statement:

```
SELECT ID FROM ACCNT WHERE ACCNT_NB = 123456
```

is replaced in the cache by:

```
SELECT ID FROM ACCNT WHERE ACCNT_NB = &
```

While better performing than full PREPAREs, dynamic SQL literal replacement has extra overhead in processing the statement text to determine a match; therefore it is not nearly as efficient as using parameter markers. Besides the criteria that the statement text must match exactly except for literal values, the literals must be of the same data type to allow a match.

Consider the three WHERE clauses in Example 7-13. They look the same but they have a different number of blanks in the statement text after the literal. For a process application like DSNTEP2, the extra blanks are stripped out and the statements match. Even though they have different lengths, when the literal is substituted with a ‘&’ then the resulting strings all match in length and kind.

Example 7-13 Dynamic SQL literal replacement

```
WHERE INTEGER_COL = 1 AND CHAR_COL = 'X'  
WHERE INTEGER_COL = 01 AND CHAR_COL = 'X'  
WHERE INTEGER_COL = 001 AND CHAR_COL = 'X'
```

But for other processes that just pass a character string, these three statements do not match because of the extra blanks. Programs like DSNTIAD and DSNTIAUL pass the text as it is written; therefore the statements do not match unless the literal strings are the same length.

To demonstrate the effectiveness of CONCENTRATE STATEMENT WITH LITERALS (abbreviated to CSWL in the figures and tables), we ran three different variations of an application test using the same statement with a different number of literals (1, 10, 20, 100). Each test runs the SQL statement 1,000 times, which smooths the results as the parameter marker test and the CONCENTRATE STATEMENT WITH LITERALS test runs do a full PREPARE on the first execution.

- ▶ The first variation is using parameter markers.
- ▶ The second is using CONCENTRATE STATEMENT WITH LITERALS.
- ▶ The third is just using the full statement text with literals (a full PREPARE on each execution).

In each case the dynamic statement cache is cleared of all statements prior to the beginning of the test.

Example 7-14 shows the SQL statement used in these tests.

Example 7-14 Sample query for testing CSWL performance improvement

```
STRING = 'SELECT COUNT(*) FROM TABLE1 WHERE COL_1 IN (00000000  
????????????????????????????????????????????????????????????  
????????????????????????????????????????????????????????????????  
????????????????????????????????????????????????????????????????  
?????????????????????????????????????????????????????????????????)';
```

The performance measurements from these tests are shown in Figure 7-15.

This test shows that the performance of CONCENTRATE STATEMENT WITH LITERALS is much better than doing full PREPAREs. It also shows how a different number of literals in the statement affects performance.

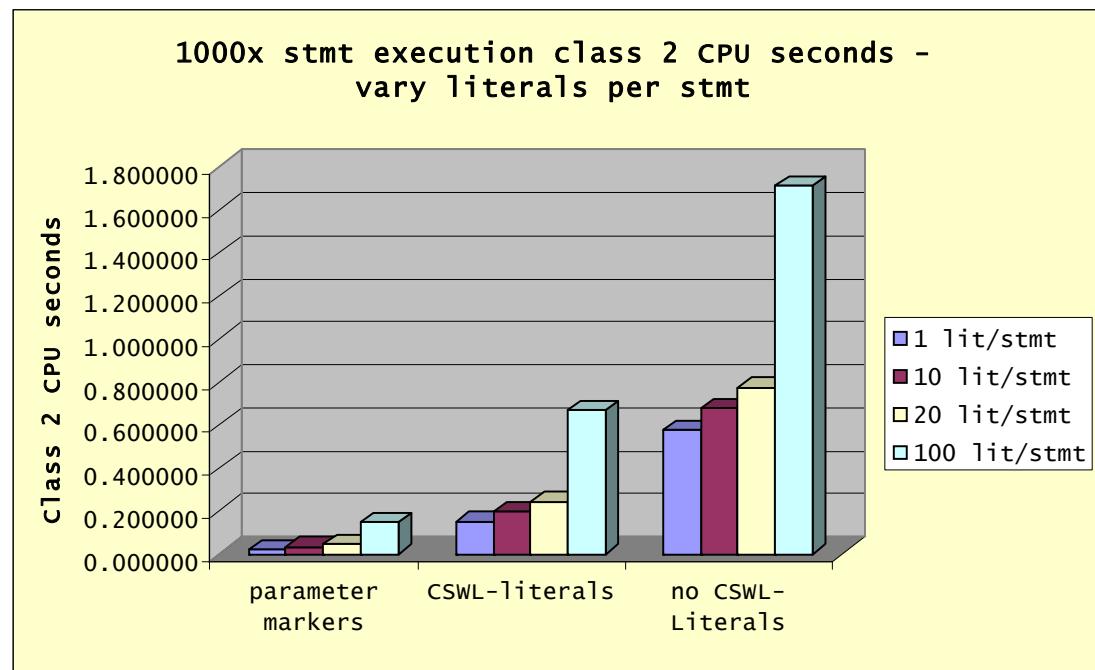


Figure 7-15 Performance measurements for dynamic SQL literal replacement

The actual numbers for class 2 CPU seconds for each test are shown in Table 7-6. You can see that the performance of CONCENTRATE STATEMENT WITH LITERALS (the set of bars in the middle) is much better than when doing full PREPAREs (the set of bars on the right). These tests also show how a different number of literals in the statement affects performance. Table 7-6 shows that the percentage improvement in CPU seconds for using CONCENTRATE STATEMENT WITH LITERALS versus doing a full PREPARE decreases as the number of literals in the SQL statement increases.

Table 7-6 Class 2 CPU time for dynamic SQL literal replacement tests

# Literals per statement	CPU in seconds			% improvement - CSWL versus PREPARE
	Parameter markers	Literals with CSWL	Literals without CSWL (PREPARE)	
1	0.019997	0.150872	0.573976	73.71
10	0.031898	0.195385	0.677576	71.16
20	0.044513	0.244199	0.775266	68.50
100	0.147647	0.668110	1.712147	60.98

The chart in Figure 7-16 shows the relative differences in the PREPARE time only.

Depending on the complexity of the SQL statement and the length of time it normally takes to execute it, the relative PREPARE and execution time can impact whether it is effective to use CONCENTRATE STATEMENT WITH LITERALS. For instance, if a query takes an hour to run, using CONCENTRATE STATEMENT WITH LITERALS does not provide much benefit relative to the total execution time.

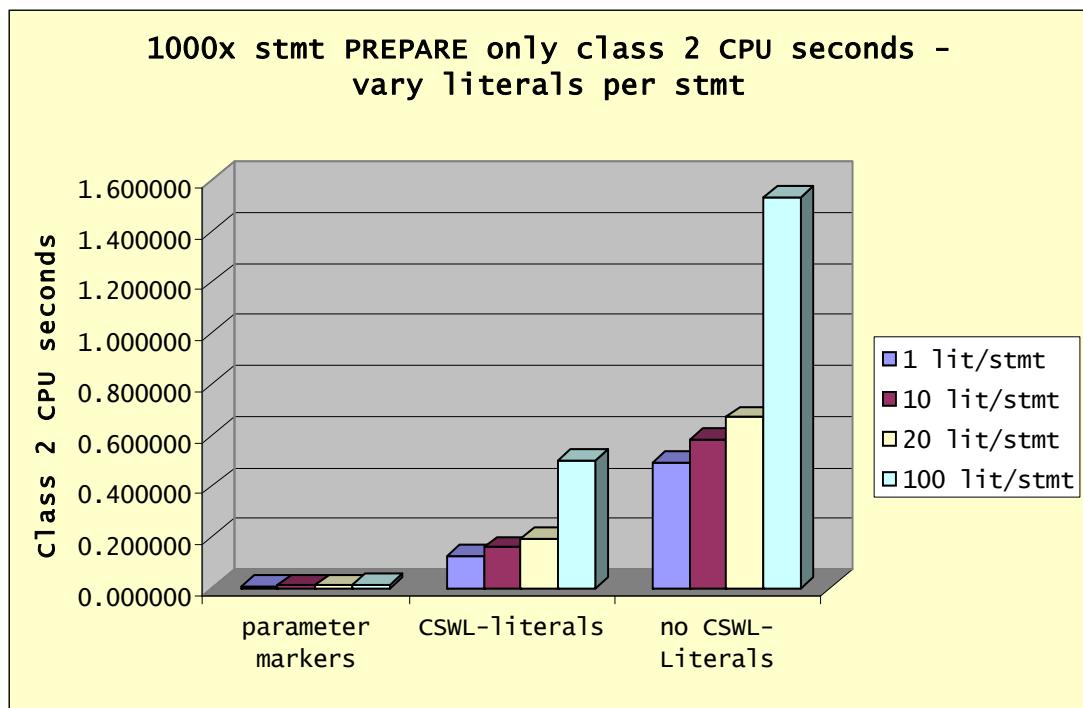


Figure 7-16 Performance measurements for dynamic SQL literal replacement - PREPARE only

The actual numbers for class 2 CPU seconds for just the PREPARE portion of each test are shown in Table 7-7.

Table 7-7 Class 2 CPU time for dynamic SQL literal replacement tests - PREPARE only

# Literals per statement	CPU time (sec)			% improvement - CSWL versus PREPARE
	Parameter markers	Literals with CSWL	Literals without CSWL (PREPARE)	
1	0.008163	0.126621	0.493197	74.33
10	0.008947	0.159018	0.584462	72.79
20	0.009614	0.193726	0.670871	71.12
100	0.015434	0.501448	1.532612	67.28

The percentage improvement in the PREPARE CPU time for using dynamic SQL literal replacement instead of doing a full PREPARE is even better than the improvement for the SQL statement as a whole. Again, the most benefit is seen when there is a smaller number of literals being replaced.

The ability to cache dynamic SQL statements with literals provided a 60% to 74% reduction in class 2 CPU times in tests compared to doing a full PREPARE. The benefit of this feature is greatest for very short running SQL statements which are run repeatedly, where the only change in each execution is the value of one or more literals. Typically, the majority of the cost for these types of SQL statements is the PREPARE cost. Because the ability to cache dynamic SQL statements with literals avoids the cost of a full PREPARE, these types of statements show the greatest reduction in CPU time.

There are changes to the DSN_STATEMENT_CACHE_TABLE to identify those cached statements which have their literals replaced with ampersands (&). There are also changes to various statistics trace records to support this enhancement. The DB2 instrumentation has been enhanced to report the exploitation of literals replacement, and you can track its use in many ways. Example 7-15 shows a portion of an OMEGAMON PE Accounting report that includes the Dynamic SQM STMT section.

Example 7-15 OMEGAMON PE report showing statistics on CSWL

1 LOCATION: DB0A	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1)					PAGE: 1-3
GROUP: N/P	ACCOUNTING TRACE - LONG					REQUESTED FROM: NOT SPECIFI
<hr/>						
DYNAMIC SQL STMT	TOTAL	DRAIN/CLAIM	TOTAL	LOGGING	TOTAL	MISCELLANEOUS
	-----	-----	-----	-----	-----	-----
REOPTIMIZATION	0	DRAIN REQST	0	LOG RECS WRITTEN	0	MAX STO LOB VAL (KB)
NOT FOUND IN CACHE	0	DRAIN FAILED	0	TOT BYTES WRITTEN	0	MAX STO XML VAL (KB)
FOUND IN CACHE	2	CLAIM REQST	12			
IMPLICIT PREPARES	0	CLAIM FAILED	0			
PREPARES AVOIDED	0					
CACHE_LIMIT_EXCEEDED	0					
PREP_STMT_PURGED	0					
CSWL - STMTS PARSED	0					
CSWL - LITS REPLACED	0					
CSWL - MATCHES FOUND	0					
CSWL - DUPLS CREATED	0					

The fields of interest for this function are as follows:

► **CSWL - STMTS PARSED**

The number of times DB2 parsed dynamic statements because CONCENTRATE STATEMENTS WITH LITERALS behavior was used for the prepare of the statement for the dynamic statement cache.

► **CSWL - LITS REPLACED**

The number of times DB2 replaced at least one literal in a dynamic statement because CONCENTRATE STATEMENTS WITH LITERALS was used for the prepare of the statement for dynamic statement cache.

► **CSWL - MATCHES FOUND**

The number of times DB2 found a matching reusable copy of a dynamic statement in cache during prepare of a statement that had literals replaced because of CONCENTRATE STATEMENTS WITH LITERALS.

► **CSWL - DUPLS CREATED**

The number of times DB2 created a duplicate STMT instance in the statement cache for a dynamic statement that had literals replaced by CONCENTRATE STATEMENTS WITH LITERALS behavior. The duplicate STMT instance was needed as a cache match failed because the literal reusability criteria was not met. For more details and information about the conditions required for getting the advantages of this technique, see *DB2 10 for z/OS Technical Overview*, SG24-7892.

Example 7-16 illustrates the syntax of the OMEGAMON PE jobs used for creating this report. See the OMEGAMON PE documentation for details.

Example 7-16 OMEGAMON PE accounting command example for CSWL

```
/*-----  
//PE      EXEC PGM=FPECMAN  
//STEPLIB DD DISP=SHR,DSN=OMEGASYS.DBOA.BASE.RKANMOD  
//INPUTDD DD    DISP=SHR,DSN=SMFDATA.DB2RECS.G5383V00  
//JOBSUMDD DD SYSOUT=*  
//SYSOUT   DD SYSOUT=*  
//ACRPTDD DD SYSOUT=*  
//UTTRCDD1 DD SYSOUT=*  
//SYSIN    DD *  
GLOBAL  
      TIMEZONE (+ 05:00)  
ACCOUNTING  
      TRACE  
      LAYOUT(LONG)  
      INCLUDE(SUBSYSTEM(DBOA))  
      INCLUDE(PRIMAUTH(DB2R1))  
EXEC  
/*
```

The statement cache table, DSN_STATEMENT_CACHE_TABLE, contains information about the SQL statements in the statement cache. This information is captured with the execution of the EXPLAIN STMTCACHE ALL statement. Example 7-17 shows an example of the execution of this command in SPUFI.

Example 7-17 EXPLAIN STMTCACHE ALL

```
-----+-----+-----+-----+-----+  
EXPLAIN STMTCACHE ALL ;  
-----+-----+-----+-----+-----+
```

DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0

You can find a sample CREATE TABLE statement for each EXPLAIN table in member DSNTESC of the SDSNSAMP library, including DSN_STATEMENT_CACHE_TABLE. The column LITERAL_REPL identifies cached statements where the literal values are replaced by the '&' symbol. Possible values are:

- ▶ R
The statement is prepared with CONCENTRATE STATEMENTS WITH LITERALS behavior and the literal constants in the statement have been replaced with '&'.
 - ▶ D
This statement is a duplicate statement instance with different literal reusability criteria.
 - ▶ blank
Literal values are not replaced.

Figure 7-17 shows an example of how you can analyze the DSN_STATEMENT_CACHE_TABLE contents using a spreadsheet software. This case illustrates the columns LITERAL REPL and STMT TEXT.

	T	U	V	W	X	BK	BL	STMT
1	LINES	LITERAL_REPLACE	PROMAINT	PROGRAM_NAME	SCHEMA	STMT_ID	STMT_TEXT	
2	5247		DB2R1	ADBMAIN	DB2R1	10	SELECT T.* FROM SYSIBM.SYSTABLES T WHERE T.NAME LIKE 'SYSTABLES%' FOR FETCH ONLY	
9	5247		DB2R1	ADBMAIN	DB2R1	11	SELECT * FROM "SYSIBM"."SYSTABLES" FOR FETCH ONLY	
10	178		DB2R1	DSNESM68	DB2R1	13	SELECT NAME, TBNAME, COLTYPE, LENGTH, NULLS, DEFAULT FROM SYSIBM.SYSCOLUMNS	
12	178		DB2R1	DSNESM68	DB2R1	12	SELECT CURRENT APPLICATION ENCODING SCHEME, INTEGER(CURRENT APPLICATION ENCODING S	
14	5247		DB2R1	ADBMAIN	DB2R1	10	SELECT T.* FROM SYSIBM.SYSTABLES T WHERE T.NAME LIKE 'SYSTABLES%' FOR FETCH ONLY	
15	1		DB2R1	SYSSH200	DB2R1	18	SELECT DSN_STATEMENT_CACHE_TABLE.BIND_CDATA, DSN_STATEMENT_CACHE_TABLE.BIND_CHC	
16	5247		DB2R1	ADBMAIN	DB2R1	15	SELECT * FROM "DB2R1"."DSN_STATEMENT_TABLE" FOR FETCH ONLY	
19	1		DB2R1	SYSSH200	DB2R1	17	SELECT DSN_STATEMENT_CACHE_TABLE.BIND_CDATA, DSN_STATEMENT_CACHE_TABLE.BIND_CHC	
22	1		DB2R1	SYSLH200	DB2R1	19	select CURRENT_CLIENT_ACCTNG, CURRENT_CLIENT_APPLNAME, CURRENT_CLIENT_USERID, CURRE	
23	5247		DB2R1	ADBMAIN	DB2R1	14	SELECT T.* FROM SYSIBM.SYSTABLES T WHERE T.NAME LIKE 'DSN_ST%' FOR FETCH ONLY	
26	5247		DB2R1	ADBMAIN	DB2R1	11	SELECT * FROM "SYSIBM"."SYSTABLES" FOR FETCH ONLY	
27	178		DB2R1	DSNESM68	DB2R1	13	SELECT NAME, TBNAME, COLTYPE, LENGTH, NULLS, DEFAULT FROM SYSIBM.SYSCOLUMNS	
28	1		DB2R1	SYSLH300	DB2R1	35	select q1."ADDRESS", q1."PASSWD", q1."USERID", q1."EMAIL", q1."CREDITCARD", q1."FULLNAME"	
29	1		DB2R1	SYSLH300	DB2R1	34	select q1."ADDRESS", q1."PASSWD", q1."USERID", q1."EMAIL", q1."CREDITCARD", q1."FULLNAME"	
30	1		DB2R1	SYSLH300	DB2R1	143	select q1."PURCHASEPRICE", q1."HOLDINGID", q1."QUANTITY", q1."PURCHASEDATE", q1."ACCO	
31	1		DB2R1	SYSLH300	DB2R1	144	select q1."PURCHASEPRICE", q1."HOLDINGID", q1."QUANTITY", q1."PURCHASEDATE", q1."ACCO	
33	178		DB2R1	DSNESM68	DB2R1	12	SELECT CURRENT APPLICATION ENCODING SCHEME, INTEGER(CURRENT APPLICATION ENCODING S	
34	5247		DB2R1	ADBMAIN	DB2R1	16	SELECT * FROM "DB2R1"."DSN_STATEMENT_CACHE_TABLE" FOR FETCH ONLY	
35	1		DB2R1	SYSLH300	DB2R1	100	UPDATE ACCOUNTEB SET CREATIONDATE = ?, OPENBALANCE = ?, LOGOUTCOUNT = ?, BALANCE = ?	
36	1		DB2R1	SYSLH300	DB2R1	89	UPDATE ACCOUNTEB SET CREATIONDATE = ?, OPENBALANCE = ?, LOGOUTCOUNT = ?, BALANCE = ?	
37	1		DB2R1	SYSLH300	DB2R1	179	select q1."LOW", q1."OPEN1", q1."VOLUME", q1."PRICE", q1."HIGH", q1."COMPANYNAME", q1	
38	1		DB2R1	SYSLH300	DB2R1	181	select q1."LOW", q1."OPEN1", q1."VOLUME", q1."PRICE", q1."HIGH", q1."COMPANYNAME", q1	
39	1		DB2R1	SYSLH300	DB2R1	70	SELECT T1.CREATIONDATE, T1.OPENBALANCE, T1.LOGOUTCOUNT, T1.BALANCE, T1.ACCOUNTID, T1	

Figure 7-17 Analysis of DSN STATEMENT CACHE TABLE and LITERAL REPLY

To build this document we cataloged the target DB2 10 for z/OS database as an ODBC data source in a workstation, then we read the data into the spreadsheet.

7.6 EXPLAIN MODE special register to explain dynamic SQL

DB2 10 provides a new special register, CURRENT EXPLAIN MODE, that can be used to turn on and off the collection of access path information for dynamic SQL statements at the application level. Within your application you can change the value of the special register by issuing the SQL statement SET CURRENT EXPLAIN MODE.

The possible values are as follows:

- ▶ NO, the default, indicates that no explain information is captured during the execution of an explainable dynamic SQL statement.
- ▶ YES indicates that explain information is captured in the explain tables as eligible dynamic SQL statements are prepared and executed.
- ▶ EXPLAIN indicates that explain information is captured in the explain tables as eligible dynamic SQL statements are prepared. However dynamic statements except SET statements, are not executed and a SQLCODE +217 is returned indicating a dynamic SQL statement was not executed.

When you issue the SET CURRENT EXPLAIN MODE statement with a value of YES or EXPLAIN, the access path information is captured, but the prepared statements are not written to the dynamic statement cache. The rationale is that the intent of this capability is the gathering of access path information and run statistics for dynamic SQL statements, not the caching and performance of dynamic SQL statements.

Do not run your applications in production with this special register set to YES because in effect it disables dynamic statement caching. Also, do not run your applications in production with this special register set to EXPLAIN because your dynamic SQL statements will not be executed. You do have the option of using a host variable in your application program to alter the value for this special register, giving you the option to have it set to NO in production and YES in test environments.

The EXPLAIN MODE special register is available in conversion mode.

7.7 Access plan stability

One of the main advantages of static SQL is that the access paths for static SQL statements are determined in advance of execution time, during the BIND process. There is, therefore, less uncertainty regarding how each SQL statement will perform because you can be aware of the access paths that were chosen, assuming that you bound your packages with EXPLAIN(YES).

However, there are instances where access paths might change without a change in the SQL statements. Query optimization in DB2 depends on many inputs, and even minor changes in the database environment can cause access paths to change. Indexes can be added or dropped. Additional statistics can be collected on tables and indexes. Maintenance can be applied that introduces DB2 optimizer changes. Migration to a new version of DB2 introduces optimizer changes.

In most cases, rebinding results in the same or an improved access path as the DB2 optimizer takes into account all of these inputs to determine the least costly access path. However, there are situations when static SQL queries are reoptimized and access paths regress. DB2 9, by APAR PK52523, introduces access plan stability support, which is the ability to save prior access paths and revert to a prior access path if you experience a regressed access path after rebinding.

DB2 10 extends access plan stability by providing the following enhancements:

- ▶ An access path repository that contains system-level access path hints and optimization options
- ▶ A new SQL statement EXPLAIN PACKAGE
- ▶ A new BIND QUERY command to populate the access path repository with hints and a new FREE QUERY to delete entries from access path repository

- ▶ A new catalog table SYSIBM.SYSPACKCOPY that contains metadata for the prior and original copies of packages
- ▶ New BIND options EXPLAIN(ONLY) and SQLERROR(CHECK)

In addition to the enhancements just listed, DB2 10 provides the following enhancements by APAR PM25679:

- ▶ A new BIND option APREUSE to allow a package to reuse access paths for static SQL statements in that package
- ▶ A new BIND option APCOMPARE to specify what actions to take if access paths change

In this section we discuss the performance implications of the access plan stability enhancements from an application environment perspective. We describe certain application development scenarios and various tolerance levels for access path changes and describe which of these enhancements you can use to address your needs. For functional details on the DB2 9 access plan stability support and the DB2 10 enhancements in this area, see *DB2 10 for z/OS Technical Overview*, SG24-7892.

7.7.1 EXPLAIN PACKAGE

There are times where you might experience a performance problem with a DB2 program, but you did not run EXPLAIN when the package was bound, so you do not have access path information in the PLAN_TABLE that you can use to troubleshoot the problem. There are many tools available to dynamically EXPLAIN the SQL statements in a program; however, those tools provide you with the access path that will be chosen if the package was bound today, not the access path that was chosen when the package was bound.

DB2 10 provides a new SQL statement, EXPLAIN PACKAGE, which externalizes to the PLAN_TABLE the access path that is currently in use for a package. There is no need to rebind the package. The EXPLAIN PACKAGE statement just makes the access path information available for querying. The access path written to the PLAN_TABLE is the one that was chosen at BIND time, not the one that will be chosen if you rebound the package today. You can use this access path information to analyze your poorly performing SQL statements in the package.

The format of the EXPLAIN PACKAGE statement is as follows:

```
EXPLAIN PACKAGE package-scope-specification
```

The syntax for *package-scope-specification* is shown in Figure 7-18.

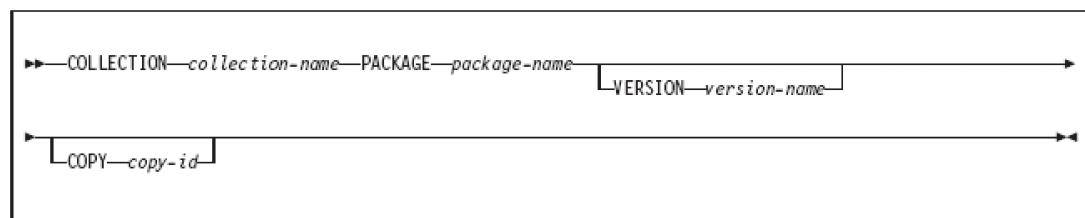


Figure 7-18 EXPLAIN PACKAGE syntax

We ran the EXPLAIN PACKAGE statement on an existing package for COBOL program APCMPCOB in collection APCMPCOB, using the sample SQL statement shown in Example 7-18.

Example 7-18 Sample EXPLAIN PACKAGE statement

```
EXPLAIN PACKAGE COLLECTION 'APCMPCOB' PACKAGE 'APCMPCOB'
```

The EXPLAIN PACKAGE statement caused an additional row to be inserted into the PLAN_TABLE. An excerpt of the new PLAN_TABLE row is shown in Table 7-8.

Table 7-8 PLAN_TABLE row for EXPLAIN PACKAGE statement

QUERYNO	PROGNAME	ACCESS TYPE	ACCESSNAME	HINT_USED
37	APCMPCOB	I	APCOMP_IX1	EXPLAIN PACKAGE: COPY 0

Note the value of “EXPLAIN PACKAGE: COPY 0” in the HINT_USED column. This tells us that PLAN_TABLE entry was generated as the result of an EXPLAIN PACKAGE statement, rather than from a BIND command. The BIND_TIME column contains the time that the BIND PACKAGE statement was executed, not when the EXPLAIN PACKAGE was executed.

The new SQL statement EXPLAIN PACKAGE can be useful for analyzing performance problems with DB2 packages because it provides you with the access path that was used for the package when it was bound, as opposed to the access path that will be used if the package is bound today. Note that most tools only provide the access path that will be used if the package is bound today.

The EXPLAIN PACKAGE statement is available in conversion mode.

7.7.2 APCOMPARE and APREUSE BIND options

Many users do not rebind packages upon migration to a new version of DB2 or upon application of DB2 maintenance for fear of access path regression. As a result, these users are not able to take advantage of new optimizations that occur after a rebind. DB2 9 provides a partial solution by allowing you to switch back to a prior version of a package if you experience access path regression. However, this solution is a reactive solution because you don't revert to a prior access path until after you incur the access path regression.

One alternative that customers have used to proactively analyze potential access path changes is to bind their packages into separate test collections upon migration to a new version of DB2 or upon application of DB2 maintenance. A second alternative is to create a separate DB2 environment that mirrors the current production environment and then bind all the packages into this new environment. Both of these alternatives are either time consuming, costly or both.

Whether or not you rebind packages is dependent upon your level of aversion to risk. Customers tend to take two approaches to risk when considering whether to rebind:

- ▶ Customers want new access paths immediately: These customers rebind packages to take advantage of new optimizations and address access path regressions as they occur.
- ▶ Customers do not want new access paths immediately: These customers are more conservative and want to know the impact of potential access path changes before rebinding. These customers typically use one of the aforementioned alternatives to analyze access paths before rebinding.

DB2 10 provides two new BIND options, APCOMPARE and APREUSE, to assist customers in analyzing potential impact to access paths prior to and during the BIND process.

Attention: DB2 10 plans to provide these two new BIND options if and when tests show that the functions are ready for use. As of the writing of this book, this functionality is not yet available. Open APAR PM25679 provides this functionality. The specifics of the implementation might change between the time this book is published and when the APAR closes. Always monitor the APAR for content and availability.

Introduction to APCOMPARE

The APCOMPARE option of the BIND PACKAGE and REBIND PACKAGE commands compares the access path of the active package with the access path being generated by the BIND or REBIND command. The action to take depends on the value specified for the APCOMPARE option. The possible values are as follows:

- ▶ NO or NONE
- ▶ WARN
- ▶ ERROR

If you specify APCOMPARE(NO) or APCOMPARE(NONE), then no action is taken; access paths are not compared. The BIND or REBIND processes as if this option was not included. APCOMPARE(NO) is the default.

If you specify APCOMPARE(WARN), and the new access path differs from the existing access path, then DB2 issues a warning message, but the package is bound or rebound. If you include the option EXPLAIN(YES), then DB2 records the new access path in the PLAN_TABLE and reports on comparison differences in the REMARKS column of the PLAN_TABLE.

If you specify APCOMPARE(ERROR), and the new access path differs from the existing access path, then DB2 issues a warning message and the package is not bound or rebound. If you include the option EXPLAIN(YES), then DB2 records the new access path in the PLAN_TABLE and reports on comparison differences in the REMARKS column of the PLAN_TABLE.

In addition, DB2 10 provides the new BIND option EXPLAIN(ONLY). If you specify EXPLAIN(ONLY), then DB2 populates the EXPLAIN tables, but does not rebind the package, regardless of your choice of option for APCOMPARE. You can use EXPLAIN(ONLY) in conjunction with APCOMPARE just to do proactive access path analysis and report on differences. If you specify EXPLAIN(ONLY), it does not matter what value you choose for APCOMPARE. DB2 does not create a new version of the package. Using EXPLAIN(ONLY) provides similar functionality to binding a package into a separate collection to analyze the access path.

Introduction to APREUSE

The APREUSE option of the BIND PACKAGE and REBIND PACKAGE commands also compares the access path of the active package with the access path being generated by the BIND or REBIND command. APREUSE is short for “Access Path Reuse” and is used to tell the optimizer whether or not you want to regenerate the package structures using the existing access paths. APREUSE can take advantage of performance improvements from rebind without changing the access path. The action to take depends on the value specified for the APREUSE option. The possible values are as follows:

- ▶ NO or NONE
- ▶ ERROR

If you specify APREUSE(NO) or APREUSE(NONE), then no action is taken; access paths are not reused. The BIND or REBIND processes as if this option was not included. APREUSE(NO) is the default.

If you specify APREUSE(ERROR), and the new access path differs from the existing access path, then DB2 issues an error message and the package is not bound or rebound. If you include the option EXPLAIN(YES), then DB2 records the new access path in the PLAN_TABLE and reports on comparison differences in the REMARKS column of the PLAN_TABLE.

You can also use EXPLAIN(ONLY) with APREUSE, in which case the package is not rebound regardless of what value is specified for the APREUSE option.

When you use the APREUSE option, the APCOMPARE option is implied, because DB2 needs to do the comparison in order to determine whether the old access paths were indeed completely reused. When both APREUSE and APCOMPARE are used, and one is specified with the value WARN and one is specified with the value ERROR, DB2 behaves as if both are specified with the value ERROR.

APCOMPARE and APREUSE test cases

Both the APCOMPARE and the APREUSE options of the BIND PACKAGE and REBIND PACKAGE commands assume that there is an existing package and that you want to compare the access paths for the existing package with the access paths generated by a new invocation of the BIND PACKAGE or REBIND PACKAGE command for the same package.

To test these features, we created the table and index described in Example 7-19.

Example 7-19 DDL to create table and index for APCOMPARE and APREUSE examples

```
CREATE TABLE APCOMP_TABLE
  (ID_NUM1  SMALLINT NOT NULL WITH DEFAULT
   ,ID_NUM2  SMALLINT NOT NULL WITH DEFAULT
   ,NAME_COL CHAR(10) NOT NULL WITH DEFAULT
   ,AGE_COL  SMALLINT NOT NULL WITH DEFAULT
   ,AP_TEXT  CHAR(200) NOT NULL WITH DEFAULT
  )
VOLATILE
;
CREATE INDEX APCOMP_IX1
ON APCOMP_TABLE
  (ID_NUM1 ASC)
;
```

We created the table as VOLATILE to ensure that DB2 will choose index access, even for a small table. We then inserted 400 rows into the table, with the distribution of values as shown in Table 7-9.

Table 7-9 Distribution of data in table for APCOMPARE and APREUSE examples

	ID_NUM1	ID_NUM2	NAME_COL	AGE_COL
first 100 rows	1	1 through 100	DOUGLAS	42
second 100 rows	2	1 through 100	JONES	42
third 100 rows	3	1 through 100	SMITH	42
fourth 100 rows	4	1 through 100	ADAMS	42

To test the capabilities of the new APCOMPARE and APREUSE BIND options, we created a simple COBOL program, named APCMPCOB, that selects a single row from the APCOMP_TABLE. Example 7-20 shows the SELECT statement executed in the program.

Example 7-20 SELECT statement for use in APCOMPARE and APREUSE examples

```
EXEC SQL
  SELECT
    ID_NUM1, ID_NUM2
    ,NAME_COL, AGE_COL
    ,AP_TEXT
  INTO
    :ID-NUM1, :ID-NUM2
    ,:NAME-COL, :AGE-COL
    ,:AP-TEXT
  FROM APCOMP_TABLE
  WHERE ID_NUM1 = 3
    AND ID_NUM2 = 97
  FETCH FIRST 1 ROW ONLY
END-EXEC.
```

We initially bound the program with the BIND option EXPLAIN(YES) to populate the PLAN_TABLE with the access path information for the SELECT statement. A summary of the Explain results is shown in Table 7-10.

Table 7-10 PLAN_TABLE contents for initial BIND of program APCMPCOB

QUERYNO	PROGNAME	ACCESS TYPE	ACCESSNAME	MATCH COLS	REMARKS
37	APCMPCOB	I	APCOMP_IX1	1	

The PLAN_TABLE shows us that DB2 chose index APCOMP_IX1, on column ID_NUM1, which is what we can expect.

Now let us create a second index on the table, one that references both the columns in our predicate. The DDL to create that index is shown in Example 7-21. After creating the second index, we ran RUNSTATS to update the catalog statistics to allow the optimizer to be aware of the statistics for both indexes.

Example 7-21 DDL to create second index for APCOMPARE and APREUSE examples

```
CREATE INDEX APCOMP_IX2
ON APCOMP_TABLE
  (ID_NUM1, ID_NUM2 ASC)
;
```

With our table and index created, our initial bind done, and a new index created that offers an alternative access path, we are now ready to test out the various combinations of APCOMPARE and APREUSE.

APCOMPARE test cases

Let us assume that we want to take a conservative approach to rebinding packages. We do not want to automatically pick up new access paths without performing some analysis first. In addition, we want to do the analysis on all packages before rebinding any packages. The method to use to accomplish this conservative approach is to rebind our packages using APCOMPARE(ERROR) and EXPLAIN(ONLY).

The REBIND command for our test program APCMPCOB is shown in Example 7-22.

Example 7-22 BIND options to only compare access paths and see what has changed

```
DSN SYSTEM(DBOA)
REBIND PACKAGE (APCMPCOB.APCMPCOB) +
APCOMPARE(ERROR) EXPLAIN(ONLY)
```

Because we bound with APCOMPARE(ERROR), we expect that the REBIND will fail if the access path changed. The output from the REBIND command is shown in Example 7-23.

Example 7-23 Output from REBIND using APCOMPARE(ERROR) and EXPLAIN(ONLY)

```
READY
DSN SYSTEM(DBOA)
DSN
REBIND PACKAGE (APCMPCOB.APCMPCOB) APCOMPARE(ERROR) EXPLAIN(ONLY)
DSNT285I -DBOA DSNTBBP2 REBIND FOR PACKAGE = DBOA.APCMPCOB.APCMPCOB,
USE OF APCOMPARE RESULTS IN:
 0 STATEMENTS WHERE COMPARISON IS SUCCESSFUL
 1 STATEMENTS WHERE COMPARISON IS NOT SUCCESSFUL
 0 STATEMENTS WHERE COMPARISON COULD NOT BE PERFORMED.
DSNT254I -DBOA DSNTBRB2 REBIND OPTIONS FOR
PACKAGE = DBOA.APCMPCOB.APCMPCOB.()
EXPLAIN ONLY
APREUSE
APCOMPARE ERROR
APRETAINUP YES
DSNT233I -DBOA UNSUCCESSFUL REBIND FOR
PACKAGE = DBOA.APCMPCOB.APCMPCOB.()
DSN
END
```

The partial output of the REBIND command shows just the options pertinent to the use of APCOMPARE. In this case we used APCOMPARE(ERROR) and EXPLAIN(ONLY). Notice that the REBIND was unsuccessful because there were statements that failed the comparison. Message DSNT285I shows how many statements were not compared successfully.

Table 7-11 shows excerpts from the PLAN_TABLE contents for the initial BIND of program APCMPCOB and for the subsequent REBIND with APCOMPARE(ERROR) and EXPLAIN(ONLY). The REMARKS column shows that the access path compare failed because the ACCESSNAME column changed. The optimizer is indicating that it makes use of the new index on both columns referenced in the WHERE clause, as indicated by a MATCHCOLS value of 2. In this case, the row in the PLAN_TABLE is not for a successfully bound plan. It is for a comparison that failed.

Table 7-11 PLAN_TABLE contents for initial BIND and for BIND with APCOMPARE(ERROR)

QUERYNO	PROGNAME	ACCESS TYPE	ACCESSNAME	MATCH COLS	REMARKS
37	APCMPCOB	I	APCOMP_IX1	1	
37	APCMPCOB	I	APCOMP_IX2	2	APCOMPARE FAILURE (COLUMN: ACCESSNAME)

You can use this process and the PLAN_TABLE data that is generated to determine what access paths will change if you rebound your packages. We might also have just used EXPLAIN(ONLY) to populate the PLAN_TABLE and not rebind the package. The difference is that, when we use EXPLAIN(ONLY) without APCOMPARE(ERROR), we get the PLAN_TABLE populated, but we do not get the REMARKS column populated with any differences.

You can use APCOMPARE(ERROR) and EXPLAIN(ONLY) as a replacement for any process you currently have in place to rebind packages into a separate collection. This new process provides you with the same functionality without creating many packages and without increasing the size of the SPT01 directory table space.

Now, let us examine the case where we want to take a moderate approach to rebinding packages. We do now want to automatically pick up new access paths without performing some analysis first. However, if the access path did not change, we want to go ahead and let the rebind succeed. The method to use to accomplish this moderate approach is to rebind our packages using APCOMPARE(ERROR) and EXPLAIN(YES). The REBIND command for our test program APCMPCOB is shown in Example 7-24.

Example 7-24 BIND options to use APCOMPARE to fail REBIND if access path has changed

```
DSN SYSTEM(DBOA)
REBIND PACKAGE (APCMPCOB.APCMPCOB) +
    APCOMPARE(ERROR) EXPLAIN(YES)
```

The APCOMPARE(ERROR) option compares the active access path with the one selected at REBIND time. If the access path did not change, then the rebind is successful. If there is a difference in access path, the rebind does not take place and, in this case, the newly calculated access path is NOT written to the PLAN_TABLE. Any time you specify APCOMPARE(ERROR), the rebind fails when the access path changes. The only time the difference in access path is recorded in the PLAN_TABLE is if you specify EXPLAIN(ONLY).

The output from our rebind with APCOMPARE(ERROR) and EXPLAIN(YES) is shown in Example 7-25.

Example 7-25 Output from REBIND using APCOMPARE(ERROR) and EXPLAIN(YES)

```
READY
    DSN SYSTEM(DBOA)
    DSN
        REBIND PACKAGE (APCMPCOB.APCMPCOB) APCOMPARE(ERROR) EXPLAIN(YES)
DSNT285I -DBOA DSNTBBP2 REBIND FOR PACKAGE = DBOA.APCMPCOB.APCMPCOB,
    USE OF APCOMPARE RESULTS IN:
        0 STATEMENTS WHERE COMPARISON IS SUCCESSFUL
        1 STATEMENTS WHERE COMPARISON IS NOT SUCCESSFUL
        0 STATEMENTS WHERE COMPARISON COULD NOT BE PERFORMED.
DSNT233I -DBOA UNSUCCESSFUL REBIND FOR
    PACKAGE = DBOA.APCMPCOB.APCMPCOB.()
    DSN
    END
```

The DSNT285I message tells us that there was a difference in the access path, therefore the rebind was not successful. No rows are written to the PLAN_TABLE in this case, even though we specified EXPLAIN(YES), because no rebind took place.

Now, let us examine the case where we want to take a more aggressive approach to rebinding packages. We want to automatically pick up new access paths without performing some analysis first. However, if the access path did change we want to be aware of it. The method to use to accomplish this more aggressive approach is to rebind our packages using APCOMPARE(WARN) and EXPLAIN(YES). The REBIND command for our test program APCMPCOB is shown in Example 7-26.

Example 7-26 BIND options to use APCOMPARE to warn us if access path has changed

```
DSN SYSTEM(DBOA)
    REBIND PACKAGE (APCMPCOB.APCMPCOB) +
        APCOMPARE(WARN) EXPLAIN(YES)
```

The APCOMPARE(WARN) option compares the active access path with the one selected at REBIND time. Regardless of whether the access path changed, the rebind is successful. If there is a difference in access path, the rebind still takes place, but you also receive a DSNT285I warning message that tells you that the access path changed.

The output from our rebind with APCOMPARE(WARN) and EXPLAIN(YES) is shown in Example 7-27.

Example 7-27 Output from REBIND using APCOMPARE(WARN) and EXPLAIN(YES)

```
READY
    DSN SYSTEM(DBOA)
    DSN
        REBIND PACKAGE (APCMPCOB.APCMPCOB) APCOMPARE(WARN) EXPLAIN(YES)
    DSNT285I -DBOA DSNTBBP2 REBIND FOR PACKAGE = DB0A.APCMPCOB.APCMPCOB,
        USE OF APCOMPARE RESULTS IN:
            0 STATEMENTS WHERE COMPARISON IS SUCCESSFUL
            1 STATEMENTS WHERE COMPARISON IS NOT SUCCESSFUL
            0 STATEMENTS WHERE COMPARISON COULD NOT BE PERFORMED.
    DSNT254I -DBOA DSNTBRB2 REBIND OPTIONS FOR
        PACKAGE = DB0A.APCMPCOB.APCMPCOB.()
        EXPLAIN      YES
        APCOMPARE   WARN
        APRETAINDUP YES
    DSNT232I -DBOA SUCCESSFUL REBIND FOR
        PACKAGE = DB0A.APCMPCOB.APCMPCOB.()
    DSN
    END
```

The partial output of the REBIND command shows just the options pertinent to the use of APCOMPARE. In this case we used APCOMPARE(WARN) and EXPLAIN(YES). Notice that the REBIND was successful even though there were statements that failed the comparison. Message DSNT285I shows how many statements were not compared successfully.

Because we specified EXPLAIN(YES) and the rebind was successful, the new access path is written to the PLAN_TABLE. Because we also specified APCOMPARE(WARN), the REMARKS column is updated with a warning about what changed. The access paths for the initial bind and for the rebind with APCOMPARE(WARN) and EXPLAIN(YES) are exactly the same as what is shown in Table 7-11 on page 225.

Another option is to specify APCOMPARE(WARN) and EXPLAIN(NO). The rebind behavior is the same as for APCOMPARE(WARN) and EXPLAIN(YES), in that you get a warning message, but the rebind succeeds. However, you do not get an entry in the PLAN_TABLE to show what the change in access path is.

You can see from these various test cases that APCOMPARE can be used to do proactive or reactive access path analysis to satisfy most levels of aversion to risk.

APREUSE test cases

Let us assume that we want to take a different approach to rebinding packages than what is available with APCOMPARE alone. We want to reuse the existing access path whenever possible, because we don't want to be subject to any possible access path regression. But we also want to take advantage of any non-access path related performance gains that come with rebinding after migrating to DB2 10. The APREUSE option can provide this capability.

The method to use to accomplish this approach is to rebind our packages using APREUSE(ERROR) and EXPLAIN(YES). The REBIND command for our test program APCMPCOB is shown in Example 7-28.

Example 7-28 BIND options to use APREUSE to rebind using the existing access path

```
DSN SYSTEM(DBOA)
      REBIND PACKAGE (APCMPCOB.APCMPCOB) +
                      APREUSE(ERROR) EXPLAIN(YES)
```

DB2 attempts to reuse the existing access path when rebinding the package. If the existing access path cannot be reused, then the rebind fails and the package is left as is.

We attempted to reuse the existing access path after adding index APCOMP_IX2. Even though this new index appears to be more beneficial than APCOMP_IX1, because it provides a two column match instead of a one column match, we want to reuse the existing access path to ensure that we experience the same performance behavior as in production today.

The output from our rebind with APREUSE(ERROR) and EXPLAIN(YES) is shown in Example 7-29.

Example 7-29 Output from REBIND using APREUSE where access path is reused

```
READY
      DSN SYSTEM(DBOA)
      DSN
      REBIND PACKAGE (APCMPCOB.APCMPCOB) APREUSE(ERROR) EXPLAIN(YES)
DSNT286I   -DBOA DSNTBPP2 REBIND FOR PACKAGE = DBOA.APCMPCOB.APCMPCOB,
      USE OF APREUSE RESULTS IN:
      1 STATEMENTS WHERE APREUSE IS SUCCESSFUL
      0 STATEMENTS WHERE APREUSE IS EITHER NOT SUCCESSFUL
          OR PARTIALLY SUCCESSFUL
      0 STATEMENTS WHERE APREUSE COULD NOT BE PERFORMED
      0 STATEMENTS WHERE APREUSE WAS SUPPRESSED BY OTHER HINTS.
```

Informational message DSNT286I is issued when APREUSE is used. The message shows us that DB2 was able to reuse the existing access path (APREUSE IS SUCCESSFUL). The access path is shown in Table 7-12.

Table 7-12 PLAN_TABLE contents for REBIND with APREUSE - access path is reused

QUERYNO	PROGNAME	ACCESS TYPE	ACCESSNAME	MATCH COLS	REMARKS
37	APCMPCOB	I	APCOMP_IX1	1	
37	APCMPCOB	I	APCOMP_IX1	1	

We specified EXPLAIN(YES), so we do write the access path to the PLAN_TABLE. Because we specified APREUSE(ERROR), and because DB2 was able to reuse the access path, the old entry in the PLAN_TABLE and the new entry are the same. This shows that DB2 was able to reuse the existing access path, even though there was an alternative access path that possibly might have been less costly. We achieved our desired result because we did not want the access path to change.

If we were not able to reuse the existing access path when specifying APREUSE(ERROR), then the rebind will fail and we will receive message DSNT292I.

If APREUSE(ERROR) is used, and DB2 is unable to reuse the old access paths, the package is not rebound. If this happens, the user can REBIND the failing packages without APREUSE. This will allow DB2 to create new access paths for these packages. This may expose these packages to access path changes, and APCOMPARE(WARN) EXPLAIN(YES) can be used to keep a record of the changes.

APCOMPARE and APREUSE impact on BIND command

APCOMPARE and APREUSE can be used with the BIND command in addition to the REBIND command. You might want to use these options when making changes to application programs and you want the ability to monitor access path changes to existing SQL statements in those programs. New and changed SQL statements have their access paths determined without regard to APCOMPARE and APREUSE values. Optimization for existing SQL statements that have not changed behave according to the values specified for the APCOMPARE and APREUSE options of the BIND command.

The access plan stability enhancements provide you with more possibilities to manage access paths and to minimize the risk of access path regression. In order to best enable customers to make use of these features, DB2 10 changes the default for the PLANMGMT option of the REBIND command from OFF to EXTENDED. If you use default values for BIND and REBIND, then you might see an increase in elapsed time and CPU time associated with these commands after migrating to DB2 10. For details on the performance cost of using the various PLANMGMT options, see 2.1.4, “BIND and REBIND stability and performance” on page 19.

The new BIND and REBIND options APCOMPARE and APREUSE provide the capability to more easily ensure access path stability when binding and rebinding packages. The two options and the different values you can specify for each, along with the new EXPLAIN(ONLY) option, provide a considerable amount of flexibility in how proactive or reactive you want to be for analyzing potential access path changes.

These options are especially useful to simplify the process of access path analysis during DB2 version migrations and during application of DB2 maintenance. Rebinding is always a good idea when you migrate to a new version of DB2, for the reasons described in 11.3.4, “Rebind during migration”. The APCOMPARE and APREUSE options can allow you to more easily rebind your programs while minimizing the chance of incurring access path regression.

You can use APCOMPARE and APREUSE even if you have never run EXPLAIN on your packages if the package was bound in a recent version of DB2.

The APCOMPARE and APREUSE bind options are available in conversion mode. The EXPLAIN(ONLY) option is also available in conversion mode.

7.8 Access currently committed data

DB2 has made some changes over the past few versions to control whether or not uncommitted data is accessed by queries.

7.8.1 Overview

Prior to DB2 9, DB2 allows access only to data that is committed and consistent, unless UNCOMMITTED READ (UR) isolation level is used, either as a BIND option or by specifying WITH UR on individual SQL statements.

DB2 9 introduces the system parameter SKIPUNCI (also made available to DB2 Version 8 by APAR) which causes applications using row level locking to ignore rows inserted by other applications, but have not yet been committed. Because SKIPUNCI is a system parameter, allowing applications to ignore uncommitted inserts will apply to all applications, which might not be what you want.

DB2 9 also allows applications to skip locked rows. The SKIP LOCKED DATA option allows a transaction to skip rows that are incompatibly locked by other transactions. Because the SKIP LOCKED DATA option skips these rows, the performance of some applications can be improved by eliminating lock wait time. However, use the SKIP LOCKED DATA option only for applications that can reasonably tolerate the absence of the skipped rows in the returned data and produce inconsistent results. The SKIP LOCKED DATA clause is specified in a SELECT, SELECT INTO, PREPARE, searched UPDATE, or searched DELETE statement. You can also use the SKIP LOCKED DATA option with the UNLOAD utility.

DB2 10 provides the capability, at an application level, to allow applications to access only data that is committed at the time of the SQL statement and to ignore data that is not yet committed. As a result, when this option is used, applications do not see rows that have been inserted but not yet committed, and those same applications see rows that have been deleted but not yet committed. The option to allow this feature can be coded as a BIND option for static SQL, in a PREPARE statement for dynamic SQL and in DDL for creating or altering functions and stored procedures and *only applies to UTS*.

The clause USE CURRENTLY COMMITTED introduced on the PREPARE statement is shown in Figure 7-19.

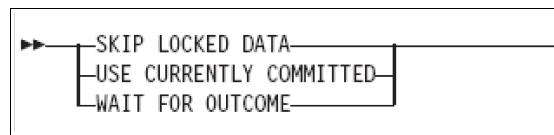


Figure 7-19 PREPARE clauses

Details on the syntax and usage of this option can be found in *DB2 10 for z/OS Technical Overview*, SG24-7892, or the DB2 manuals.

The impact on performance for this feature is directly related to the cost of acquiring locks and the cost of determining the state of the row, either committed or not committed. We ran a number of performance measurements comparing different combinations of options. For the purposes of these tests, the following conditions were in effect:

- ▶ Baseline tests:
 - All INSERT and DELETE statements were committed prior to SELECT statements being issued by another application. This eliminated the possibility of time-outs.
 - SKIPUNCI DSNZPARM is set to NO.
 - Isolation level is cursor stability (CS).
 - Tests were run for row level locking and page level locking.
- ▶ New feature tests:
 - INSERT and DELETE statements were not committed prior to SELECT statements; therefore SELECT statements are reading uncommitted data.
 - SKIPUNCI DSNZPARM is set to NO.
 - Tests were run for Isolation level UR versus new feature with isolation level CS.
 - Tests were run for row level locking and page level locking.
- ▶ SKIPUNCI tests:
 - SKIPUNCI DSNZPARM is set to YES.
 - Test was run for row level locking only.
 - Isolation level is cursor stability (CS).

The workloads being measured were defined as shown in Table 7-13.

Table 7-13 Workload descriptions for access currently committed data tests

Test scenarios	Workload ID	Test case description
Skip uncommitted INSERT	10	10 row transaction
	100	100 row transaction
	500	500 row transaction
Skip uncommitted DELETE	20	DELETE 1 row, SELECT 20 rows (10 on either side of deleted row)
	100A	DELETE 10 rows, SELECT 100 rows
	100B	DELETE 20 rows, SELECT 200 rows

7.8.2 Measurements

Seven sets of measurements are documented in the following sections:

- ▶ Committed inserts versus uncommitted inserts: Row level locking
- ▶ Committed inserts versus uncommitted inserts: Page level locking
- ▶ Skip uncommitted inserts: DSNZPARM versus application: Row level locking
- ▶ Access currently committed versus WITH UR
- ▶ Select uncommitted DELETE: Row level locking
- ▶ Select uncommitted DELETE: Page level locking
- ▶ Wait for commit versus skip uncommitted insert

Committed inserts versus uncommitted inserts: Row level locking

The chart in Figure 7-20 shows the class 1 elapsed and CPU times for the baseline test and the new feature test using row level locking. This chart just shows the test of skipping uncommitted inserts.

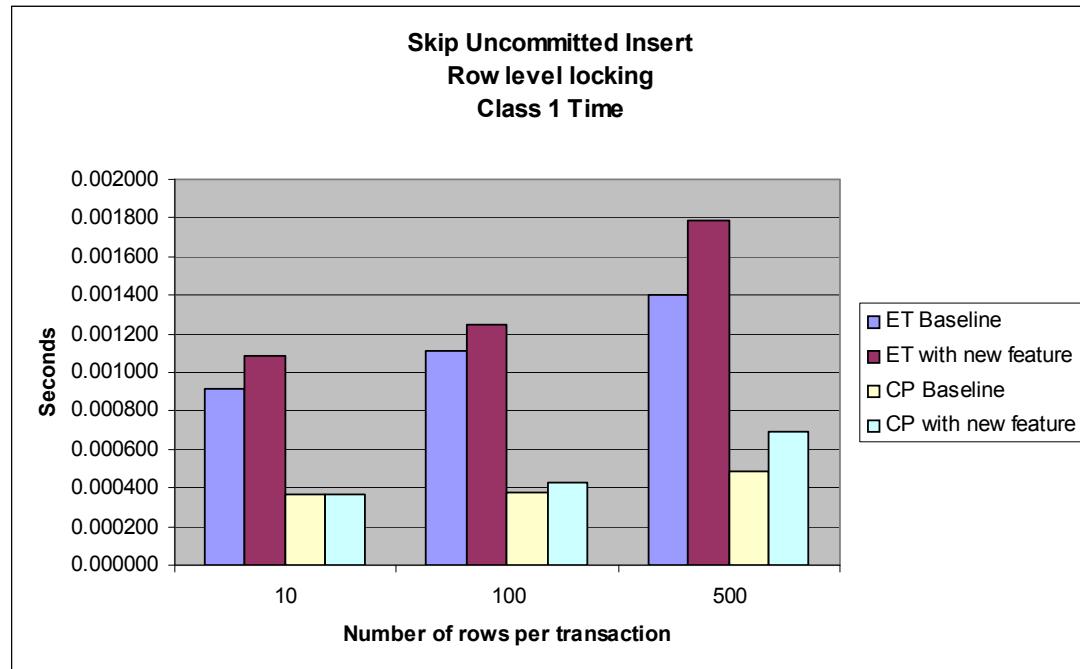


Figure 7-20 Class 1 times for skip uncommitted inserts - Row level locking

Note that the SELECT statements in the baseline test are not executed until after all the INSERT statements are committed. Therefore, there is no lock suspension time for the SELECT statements in the baseline test. The class 1 elapsed time and CPU time for the new feature are between 10% and 25% more due to the cost to check the state of each row as to whether it is committed or not. Later on we show some tests where the SELECT statements in the baseline test occur before the COMMIT and are subject to time-outs.

The greater the number of rows in the transaction, the greater the number of lock requests; therefore the CPU overhead is greater for the 500 transaction case than the 10 or 100 transaction case.

The chart in Figure 7-21 shows the class 2 elapsed and CPU times for the baseline test and the new feature test using row level locking. This chart just shows the test of skipping uncommitted inserts.



Figure 7-21 Class 2 times for skip uncommitted inserts - Row level locking

The class 2 elapsed time and CPU time for the new feature are between 10% and 25% more due to the cost to check the state of each row as to whether it is committed or not. Later on we show some tests where the SELECT statements in the baseline test occur before the COMMIT and are subject to time-outs.

The greater the number of rows in the transaction, the greater the number of lock requests; therefore the CPU overhead is greater for the 500 transaction case (almost a 50% increase) than the 10 or 100 transaction case.

Committed inserts versus uncommitted inserts: Page level locking

The chart in Table 7-14 shows the class 1 and 2 elapsed and CPU times in seconds for the baseline test and the new feature test using page level locking. This chart just shows the test of skipping uncommitted inserts. The base test is using DSNZPARM SKIPUNCI for this comparison.

Table 7-14 Class 1 and 2 times for skip uncommitted inserts - Page level locking

	10 row transaction		100 row transaction		500 row transaction	
	Base	With feature	Base	With feature	Base	With feature
Class 1 ET	0.001084	1.53901	0.001036	2.246116	0.002242	1.040001
Class 2 ET	0.001053	1.53897	0.001005	2.246078	0.002212	1.039966
Class 1 CPU	0.000367	0.000451	0.000377	0.000504	0.000487	0.000591
Class 2 CPU	0.000337	0.000414	0.000346	0.000466	0.000457	0.000557
Time-outs	0	48	0	70	0	32

The class 1 and 2 elapsed times are considerably more for the new feature with page level locking because you are more likely to experience time-outs when the number of rows in a page is greater than 8.

The class 1 and 2 CPU times for the new feature are between 20% and 30% more due to the cost to check the state of each row as to whether it is committed or not. Later on we show some tests where the SELECT statements in the baseline test occur before the COMMIT and are subject to time-outs.

With page level locking there is not a huge increase in the CPU overhead as the number of rows processed increases, because the number of locks is not as dependent on the number of rows. Instead it is dependent on the number of pages.

Skip uncommitted inserts: DSNZPARM versus application: Row level locking

The measurements in Table 7-15 show the class 1 and 2 elapsed and CPU times for the SKIPUNCI test and the new feature test using row level locking. This chart just shows the test of skipping uncommitted inserts.

Table 7-15 Class 1 and 2 times for SKIPUNCI versus new feature - Row level locking

	10 row transaction		100 row transaction	
	With feature	SKIPUNCI	With feature	SKIPUNCI
Class 1 ET	0.001084	0.266125	0.001252	0.25969
Class 2 ET	0.001053	0.266609	0.001221	0.25966
Class 1 CPU	0.000367	0.000383	0.000425	0.000398
Class 2 CPU	0.000337	0.000352	0.000395	0.000366
Time-outs		2		2

The class 1 and 2 elapsed times are considerably more for the SKIPUNCI case because of time-outs.

The class 1 and 2 CPU times for the new feature are very similar to the times for the SKIPUNCI test (less than +/- 10%). These numbers show that it is not more expensive to use the access currently committed feature at the application level than it is to control this capability at the system level. Controlling the feature at the application level gives you more flexibility as to where and when to use it.

Access currently committed versus WITH UR

The chart in Figure 7-22 shows the class 1 and 2 CPU times for the new feature test versus the use of WITH UR on the same SQL statements. In this case the “With feature” test is testing only skipping INSERT statements (no DELETE issued) and row level locking was used. The WITH UR capability was set by setting the following Java property:

```
conn1.setTransactionIsolation(Connection.TRANSACTION_READ_UNCOMMITTED);
```

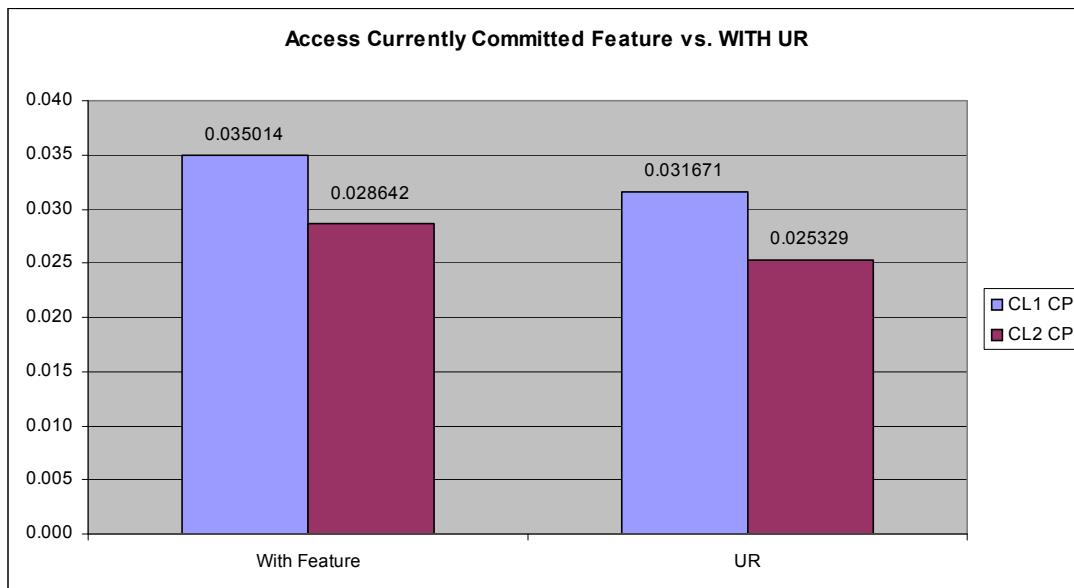


Figure 7-22 Class 1 and 2 CPU times - Currently committed versus WITH UR

The WITH UR semantic utilizes less CPU than the currently committed semantic due to less lock requests. However, WITH UR does return uncommitted rows while currently committed does not.

Select uncommitted DELETE: Row level locking

The chart in Figure 7-23 shows the class 1 and 2 CPU times for the skip uncommitted delete test versus the base test, in which the DELETE statements are committed before the SELECT statements are issued. This chart shows the test of selecting rows that have been deleted but the DELETEs have not yet been committed. The tests are as described in the second set of scenarios in Table 7-13 on page 231. In this case row level locking was used.

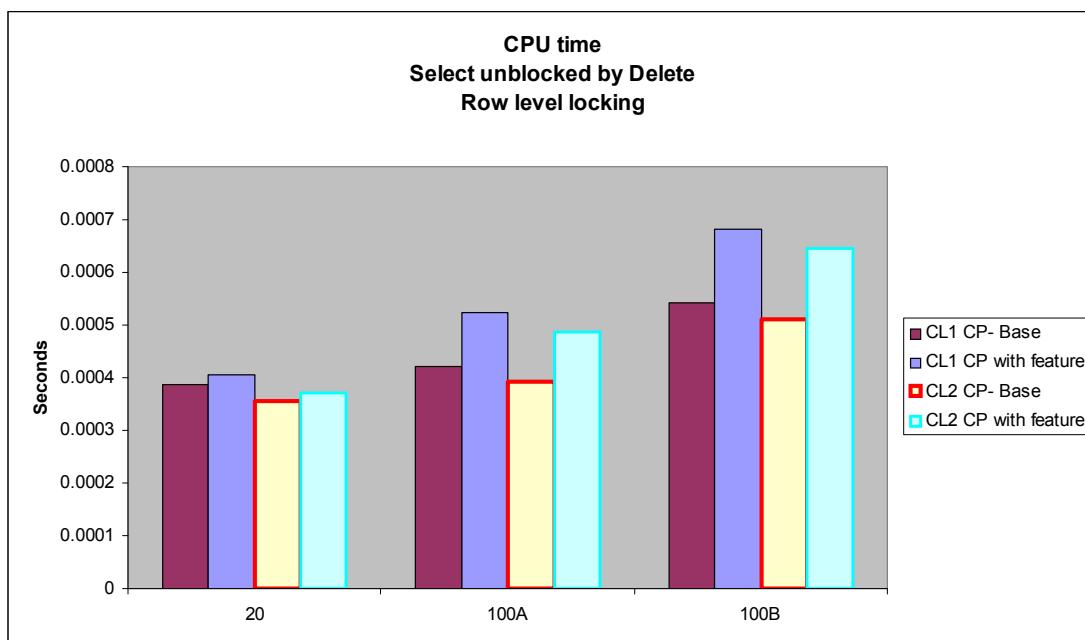


Figure 7-23 CPU times for SELECT unblocked by DELETE - Row level locking

Note that the CPU increase is even more noticeable for DELETE than it is for INSERT when this feature is used.

The chart in Figure 7-24 shows the class 1 and 2 elapsed times for the base tests versus the tests of the new feature with row level locking. The tests are for cases where rows are being deleted and are seen by the access currently committed cases. See Table 7-13 for a description of each workload that was tested.

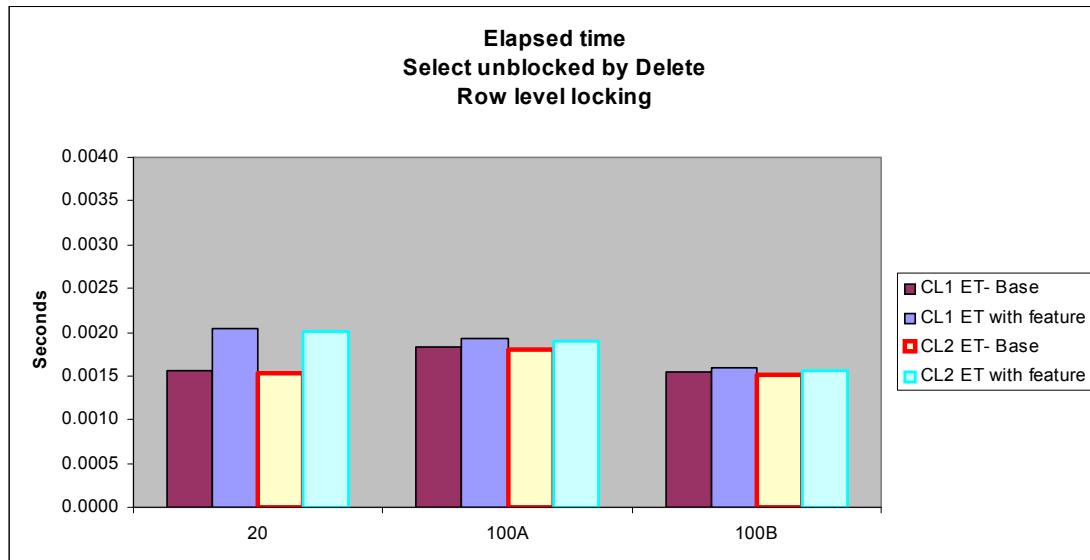


Figure 7-24 Elapsed time for SELECT unblocked by DELETE - Row level locking

The elapsed time difference per transaction is greater in the case where there is a smaller number of rows being deleted. As the number of uncommitted deletes that are read increases, the elapsed time per transaction decreases slightly.

Select uncommitted DELETE: Page level locking

The chart in Figure 7-25 shows the class 1 and 2 CPU times for the skip uncommitted delete test versus the base test, in which the DELETE statements are committed before the SELECT statements are issued. This chart shows the test of selecting rows that have been deleted but the DELETEs have not yet been committed. The tests are as described in the second set of scenarios in Table 7-13 on page 231. In this case row level locking was used.

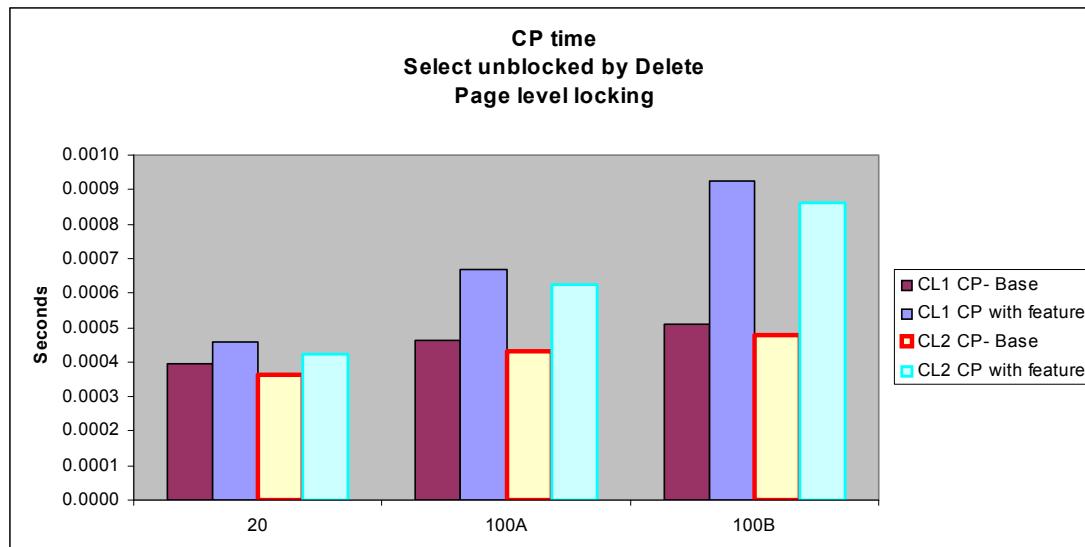


Figure 7-25 CPU times for *SELECT unblocked DELETE - Page level locking*

Note that the CPU increase is even more noticeable for DELETE when page level locking is used than when row level locking is used. In this case DB2 has to evaluate potentially more uncommitted deleted rows.

The measurements in Table 7-16 show the class 1 and 2 elapsed times for the base tests versus the tests of the new feature with page level locking. The tests are for cases where rows are being deleted and are seen by the access currently committed cases.

Table 7-16 Elapsed times for select unblocked delete - Page level locking

Test case	20		100A		100B	
	Base	With feature	Base	With feature	Base	With feature
CL1 ET	0.002009	0.002862	0.017252	2.044859	0.041688	7.790357
CL2 ET	0.001976	0.002825	0.017219	2.044816	0.041655	7.790296
Time-outs		0		32		117

Note that with page level locking there is the chance for time-outs, even with the new feature. These time-outs therefore negatively impact the total elapsed time.

Wait for commit versus skip uncommitted insert

Up to this point our measurements for the base test have been for cases where the inserts or deletes have been committed prior to any attempt to select the data. Therefore, the cost of the select is very little due to the locks being released at commit point prior to the select being executed. The advantage of the access currently committed feature is that we can read data and skip through the locks to read just data that is currently committed.

In order to do a complete “apples to apples” comparison of the cost of access currently committed, we ran a comparison where the base test attempted to read data as it was being inserted. As a result, the selects had to wait and, in some cases, even timed out. Table 7-17 shows the class 1 and 2 elapsed and CPU times for this comparison.

Table 7-17 Class 1 and 2 times - Wait for commit versus skip uncommitted INSERT

Times	Base - wait for commit	Feature - skip uncommitted INSERT
Class 1 Elapse Time	144.1608	0.113262
Class 2 Elapsed Time	144.0738	0.113218
Class 1 CPU Time	0.007587	0.001115
Class 2 CPU Time	0.006791	0.001108

You can see that elapsed times and CPU times for the access currently committed feature are magnitudes less in this test. The reason for the high cost in the base test is that the selects now have to wait for the locks to be released at commit time. In some cases the elapsed times get inflated because the transaction times out waiting for locks. This is the real world case that this new feature attempts to address.

The access currently committed feature was introduced to allow applications to skip locked rows, both inserts and deletes, at the application level instead of just at the system level. The intent is to allow you to return rows quickly without waiting for locks. You do not have the danger of returning uncommitted inserts as you do with the use of WITH UR, but you can return rows that have subsequently been deleted.

There is a CPU cost to check whether inserted rows have been committed or not, so you can see an increase in CPU time, and elapsed time, for this new feature. However, your queries are less likely to time out because you are skipping rows rather than waiting for locks to be released; as a result, your CPU time and elapsed time might be considerably less.

Not all applications can take advantage of this feature. Consider what the needs of the application are and whether or not you need to see rows that are in the process of being inserted or deleted.

This feature uses more CPU than the WITH UR function, but it only returns data that is committed at the time the query is run.

Access to currently committed data is an enhancement that is available in new function mode.



Distributed environment

DB2 10 for z/OS provides a number of enhancements to improve the availability of distributed applications, including online CDB changes and online changes to location alias names in addition to connectivity improvements.

In this chapter, we provide an introduction and a performance discussion, as well as our observations on the following topics:

- ▶ High performance DBATs
- ▶ Limited block fetch extended to the JCC Type 2 drivers
- ▶ Return to client result sets
- ▶ Enhanced support for native SQL procedures
- ▶ Extended correlation token
- ▶ Virtual and real storage with distributed IRWW workload
- ▶ LOBs and XML materialization avoidance

You can find functional details of these topics in the DB2 documentation and in *DB2 10 for z/OS Technical Overview*, SG24-7892.

In this chapter and throughout the book we use OMEGAMON PE as a reference to the product IBM Tivoli OMEGAMON XE for DB2 Performance Expert Version V5R1.

8.1 High performance DBATs

Prior to DB2 10, the RELEASE option of BIND PACKAGE is not honored in distributed applications. Packages are always de-allocated at commit, mainly to allow more availability for functions such as DDL, utilities, and BIND packages, which can be impacted if the packages were released only at the application end. The performance analysis of inactive connection processing reported that a large CPU cost occurred in package allocation and de-allocation. A CPU reduction can occur when pooling database access threads (DBAT) and then associating a pooled DBAT with a connection¹.

DB2 10 includes the following enhancements:

- ▶ CPU reduction of inactive connection processing.
- ▶ An easy method to switch between RELEASE (COMMIT) and RELEASE (DEALLOCATE) BIND options for existing distributed applications without having to rebind, so that activities such as running DDL and utilities, can happen without an outage.

DB2 10 DRDA DBATs are allowed to run accessing data under the BIND RELEASE option of the package. If a package that is associated with a distributed application is bound with RELEASE (DEALLOCATE), the copy of the package is allocated to the DBAT up until the DBAT is terminated. The DBATs hold package allocation locks even while they are not being used for client unit-of-work processing.

However, to minimize the number of different packages that can possibly be allocated by any one DBAT, distributed data facility (DDF) does not pool the DBAT and disassociates it from its connection after the unit-of-work is ended. After the unit-of-work is ended, DDF cuts an accounting record and deletes the WLM enclave, as done for inactive connection processing. Thus, the client application that requested the connection holds onto its DBAT, and only the packages that are required to run the application accumulate allocation locks against the DBAT.

If one package among many is bound with RELEASE(DEALLOCATE), then the DBAT becomes a high performance DBAT, provided that it meets the requirements. Also, similar to inactive connection processing, the DBAT is terminated after 200 (not user changeable) units-of-work are processed by the DBAT. The connection at this point is made inactive. On the next request to start a unit-of-work by the connection, a new DBAT is created or a pooled DBAT is assigned to process the unit-of-work. Normal idle thread time-out detection is applied to these DBATs.

If the DBAT is in flight processing a unit-of-work and if it has not received the next message from a client, DDF cancels the DBAT after the IDTHTOIN² value has expired. However, if the DBAT is sitting idle, having completed a unit-of-work for the connection, and if it has not received a request from the client, then the DBAT is terminated (not cancelled) after POOLINAC³ time expires.

¹ There are two modes of running distributed threads: active, in which every connection is a database access thread (DBAT) even when waiting for new client transactions up until it is disconnected, and inactive, in which DBATs are pooled and handed out to connections as needed.

² The IDTHTOIN subsystem parameter (IDLE THREAD TIMEOUT field) controls the amount of time, in seconds, that an active server thread is to be allowed to remain idle.

³ The POOLINAC subsystem parameter (POOL THREAD TIMEOUT field) specifies the approximate time, in seconds, that a database access thread (DBAT) can remain idle in the pool before it is terminated.

The honoring of package bind options for distributed application threads is available only under the following conditions:

- ▶ KEEPDYNAMIC YES is not enabled.
- ▶ CURSOR WITH HOLD is not enabled.
- ▶ CMTSTAT is set to INACTIVE.

Because RELEASE(DEALLOCATE) for distributed applications avoids pooling of DBATs, you might need to increase the MAXDBAT subsystem parameter to avoid queuing of distributed requests. The DB2 10 for z/OS storage relief below the bar (see 5.2, “Virtual and real storage” on page 134) makes this requirement addressable.

High performance DBATs and RELEASE(DEALLOCATE) do not relax the need to identify and fix bad behaving distributed applications. An example are applications that keep resources allocated across COMMITs preventing them to become INACTIVE. For details, see *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01.

8.1.1 High Performance DBATs and RELEASE(DEALLOCATE)

Starting in DB2 10, by default, DB2 honors the RELEASE bind option for database access threads, which minimizes the use of CPU resources for package allocation and deallocation processing, resulting in performance improvements. You can modify this behavior by using the new MODIFY DDF PKGREL command. (In releases prior to DB2 10, the RELEASE bind option had no effect on database access threads.)

The MODIFY DDF PKGREL BNDOPT command

DB2 10 supports the MODIFY DDF PKGREL command. You can use this command to switch on or off DB2 10 RELEASE(DEALLOCATE) behavior for distributed applications.

Example 8-1 shows the result of the -DIS DDF command. The message DSNL106I shows, in this case, the value COMMIT for the option PKGREL.

Example 8-1 -DIS DDF command reporting the PKGREL option

```
DSNL080I -DBOA DSNLTDDF DISPLAY DDF REPORT FOLLOWS:  
DSNL081I STATUS=STARTD  
DSNL082I LOCATION      LUNAME        GENERICLU  
DSNL083I DBOA          USIBMSC.SCPDBOA -NONE  
DSNL084I TCPPORT=38360 SECPORT=38362 RESPOR=38361 IPNAME=-NONE  
DSNL085I IPADDR=:10.50.1.1  
DSNL086I SQL    DOMAIN=some.domain.com  
DSNL087I ALIAS          PORT  SECPORT STATUS  
DSNL088I ABC            0      0      CANCLD  
DSNL088I TEST           0      0      STARTD  
DSNL088I TEST2          0      0      STOPD  
DSNL105I CURRENT DDF OPTIONS ARE:  
DSNL106I PKGREL = COMMIT  
DSNL099I DSNLTDDF DISPLAY DDF REPORT COMPLETE
```

DB2 10 adds the option PKGREL to the -MODIFY DDF command. This option specifies whether DB2 honors the bind options of packages that are used for remote client processing. Figure 8-1 shows its syntax.

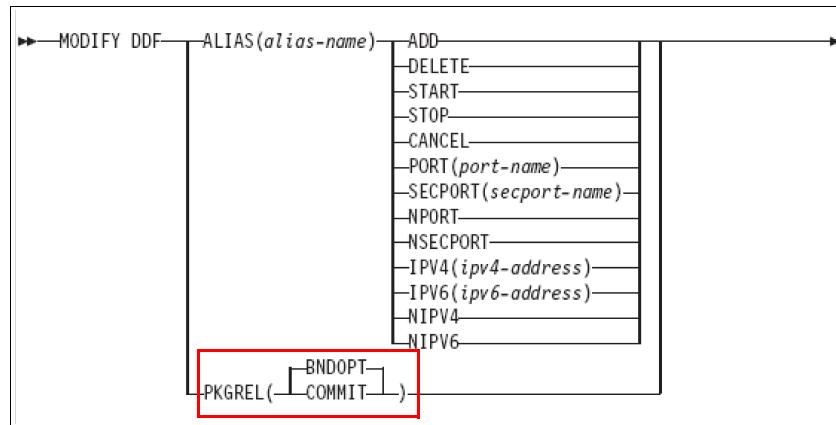


Figure 8-1 MODIFY DDF PKGREL syntax

The allowed values for the PKGREL option are as follows:

- ▶ BNDOPT: The rules of the RELEASE bind option that was specified when the package was bound are applied to any package that is used for remote client processing. BNDOPT is the default value of the MODIFY DDF PKGREL command.
- ▶ COMMIT: The rules of the RELEASE(COMMIT) bind option are applied to *any* package that is used for remote client processing.

Example 8-2 and Example 8-3 show the execution results of the MODIFY DDF PKGREL command.

Example 8-2 MODIFY DDF PKGREL(BNDOPT) output

```
21.06.55 STC12396  DSNL300I  -DBOA DSNLTMDF MODIFY DDF REPORT FOLLOWS: 936
936              DSNL302I  PKGREL IS SET TO BNDOPT
936              DSNL301I  DSNLTMDF MODIFY DDF REPORT COMPLETE
```

Example 8-3 MODIFY DDF PKGREL(COMMIT) output

```
21.06.26 STC12396  DSNL300I  -DBOA DSNLTMDF MODIFY DDF REPORT FOLLOWS: 934
934              DSNL302I  PKGREL IS SET TO COMMIT
934              DSNL301I  DSNLTMDF MODIFY DDF REPORT COMPLETE
```

PKGREL(COMMIT) is the default value when the CMTSTAT subsystem parameter is set to ACTIVE. If the MODIFY DDF PKGREL command had never been issued, then COMMIT is the default value and the CMTSTAT subsystem parameter is set to INACTIVE.

As a guideline, you can specify that DDF uses the PKGREL(BNDOPT) option during normal production operating hours. This option offers improved performance by reducing the CPU cost for allocating and deallocating packages if the packages were bound using RELEASE(DEALLOCATE). You can specify the PKGREL(COMMIT) option during routine and emergency maintenance periods where BINDs, DDL, or utilities might be executed.

Delayed effects of PKGREL(COMMIT) option

When you issue the MODIFY DDF command and specify the PKGREL(COMMIT) option, the effects are not immediate. After the command is issued, any database access thread that was running RELEASE(DEALLOCATE) packages is terminated when the connection becomes inactive.

At the next unit-of-work from the client, a new database access thread is created in RELEASE(COMMIT) mode. Any database access thread that remains active waiting for a new unit-of-work request from its client because of the rules of RELEASE(DEALLOCATE) is terminated by the DDF service task that runs every two minutes. Consequently, within approximately two minutes all database access threads run under the rules of the RELEASE(COMMIT) bind option.

Important: There is no notification about the effective application of RELEASE(COMMIT) on threads; there is no system or DB2 message indicating that all the DBATs are effectively working on PKGREL(COMMIT) mode.

RELEASE(DEALLOCATE) bind option

Consider binding a new set of distributed packages with the RELEASE(DEALLOCATE) option in order to get better control on the introduction of this feature, or to gradually migrate application into RELEASE(DEALLOCATE).

Depending on the type of statement executed by the application, DB2 uses a particular package. For example, when using dynamic placeholders, DB2 uses a package with the following naming convention: SYSSHxxy.

This naming convention can be depicted as follows:

- ▶ SYS: Package name prefix
- ▶ S: Represents a small package (65 sections); alternative value is L, indicating a large package (385 sections)
- ▶ H: Represents WITH HOLD, alternatively, this position can contain an N, indicating NOT WITH HOLD
- ▶ x: Indicates the isolation level, as follows:
 - 1=UR
 - 2=CS
 - 3=RS
 - 4=RR
- ▶ yy: The package iteration 00 through FF

So, the package SYSSH200 is a small package, WITH HOLD using isolation level CS iteration 00. By default, DB2 creates three packages for each type of package, and iteration starts with 00.

Example 8-4 shows how you can create a copy of this package in a new collection, DRDADEALLOC in this case, using the bind option RELEASE(DEALLOCATE). Note that the bind option KEEPDYNAMIC(N) is required for this package to be eligible for a High Performance DBAT.

Example 8-4 BIND COPY command

```
BIND PACKAGE(DRDADEALLOC)
  QUAL(DB2R1)
  OWNER(DB2R1)
  COPY(NULLID.SYSSH200)
```

```

SQLERROR(NOPACKAGE)
VALID(R)
ISOL(CS)
REL(D)
EXPL(NO)
CURRENTD(N)
ACTION(REPLACE)
DEGREE(1)
DYNAMICRULES(RUN)
KEEPDYNAMIC(N)
REOPT(NONE)
ENCODING(      37)
IMMEDWRITE(N)
ROUNDING(HALFEVEN)

```

The results of this BIND command can be seen in DB2 Administration Tool as shown in Example 8-5. This screen allows to see the 2 collections and the RELEASE option used during BIND.

Example 8-5 DB2 Administration Tool view of packages

Commands: BIND REBIND FREE VERSIONS GRANT ALL PLANMGMT							Scroll ==> CSR
S	Collection	Name	Owner	Bind Timestamp	V I V O Quali-	R E D	
*	*	*	*	*	* * * * *	* * *	
<hr/>							<hr/>
	DRDADEALLOC	SYSSH200	DB2R1	2011-02-25-15.01	R S Y Y DB2R1	D N R	
	NULLID	SYSSH200	DB2R1	2011-02-22-20.35	R S Y Y DB2R1	C N R	
***** END OF DB2 DATA *****							<hr/>

There are many ways of influencing the package collection used. One is the exploitation of CURRENT PACKAGE PATH; it specifies a value that identifies the path used to resolve references to packages that are used to execute SQL statements. This special register applies to both static and dynamic statements.

The value can be an empty or blank string, or a list of one or more collection IDs, where the collection IDs are enclosed in double quotation marks and separated by commas. Any quotation marks within the string are repeated as they are in any delimited identifier. The delimiters and commas are included in the length of the special register.

When CURRENT PACKAGE PATH or CURRENT PACKAGESET is set, DB2 uses the values in these registers to resolve the collection for a package. The value of CURRENT PACKAGE PATH takes priority over CURRENT PACKAGESET. In a distributed environment, the value of CURRENT PACKAGE PATH at the remote server takes precedence of the value of CURRENT PACKAGE PATH at the local server (the requester).

For example, in an application that is using SQLJ packages in collection DRDADEALLOC and a JDBC package in DB2JAVA, set the CURRENT PACKAGE PATH special register to check SQLJ1 first followed by DB2JAVA as shown in Example 8-6.

Example 8-6 SET CURRENT PACKAGE PATH

```
SET CURRENT PACKAGE PATH = DRDADEALLOC, DB2JAVA;
```

An existing SQLJ application can be customized and bound to use a specific collection using the db2sqljcustomize example shown in Example 8-7.

Example 8-7 db2sqljcustomize and -collection parameter

```
db2sqljcustomize -url jdbc:db2://system1.sv1.ibm.com:8000/ZOS1  
-user user01 -password mypass  
-rootPkgName WRKSQLJ  
-qualifier WRK1  
-collection DRDADEALLOC  
-bindoptions "CURRENTDATA NO QUALIFIER WRK1 "  
-staticpositioned YES WrkTraceTest_SJProfile0.ser
```

For more information about package resolution, see *DB2 10 for z/OS Application Programming and SQL Guide*, SC19-2969.

You can use the DB2 Configuration Assistant, a GUI tool that is shipped free of charge with the DB2 Client, for binding the DB2 distributed packages used by the DB2 Clients and the packages used by DB2 when executing dynamic SQL. This tool provides the option of selecting the RELEASE option. You can get into the Bind panel, as shown in Figure 8-2, by using the **Bind** contextual option of the Configuration Assistant. Click the **Add** button to get access to the **Add Bind Option** panel.

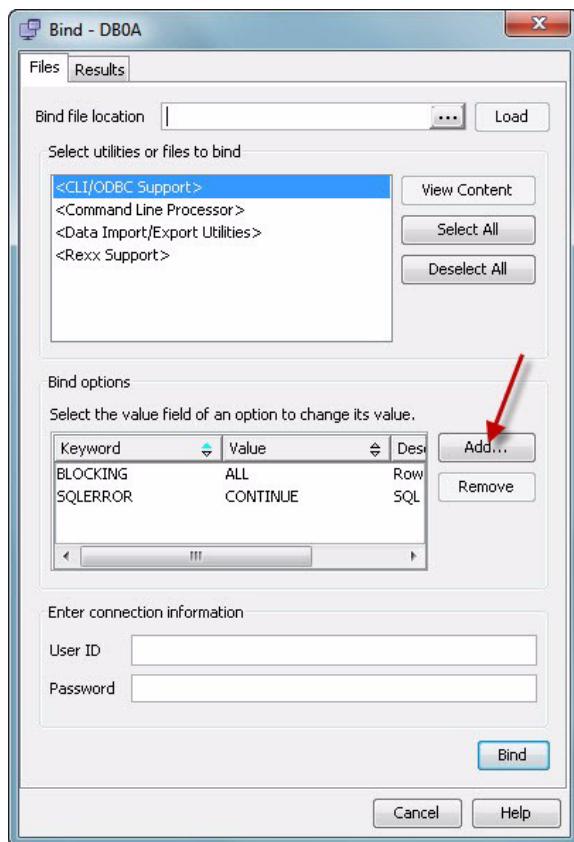


Figure 8-2 DB2 Configuration Assistant, Bind panel

Drill down until you find the **RELEASE** option. The contextual lower panel shows the options **COMMIT** and **DEALLOCATE**, as illustrated in Figure 8-3.

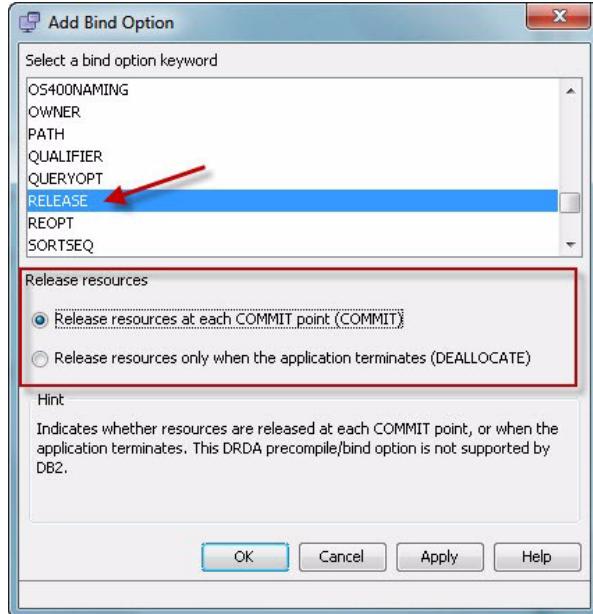


Figure 8-3 DB2 Configuration Assistant, Add Bind Option panel

DB2 Client 9.7 Fixpack 3a is required to exploit the full potential of distributed access to DB2 10 for z/OS.

Important:

- ▶ DB2 Client 9.7 Fixpack 3a changes the default from RELEASE(COMMIT) to RELEASE(DEALLOCATE).
- ▶ In addition, the related stored procedures, triggers and UDF packages can be bound with RELEASE(DEALLOCATE) to get more performance benefit.

8.1.2 High performance DBAT and RELEASE(DEALLOCATE) performance

Several test were conducted in order to report the performance advantages provided by high performance DBATs combined with RELEASE(DEALLOCATE). The workload consisted of 3 query transactions from the IBM IRWW workload (Order Status, Price Quote, Stock Level).

The test environment was:

- ▶ z10 z/OS LPAR: 3 CPs, 32 GB, z/OS 1.11
- ▶ z Linux LPAR: 2 CP
- ▶ DB2 Connect V9.7 Fix Pack 3A
- ▶ T4 JCC driver 3.59.52, JDK 1.6
- ▶ HiperSockets communication between the zLinux and the z/OS LPARs.

The tests were executed using the Auto Commit settings for the application and the packages were bound using the RELEASE(DEALLOCATE) option in a DB2 10 subsystem.

The first test execution used the PKGREL = COMMIT DDF options that provoked each package to be released at COMMIT time. The second test was ran using PKGREL = BNDOPT, as the packages were bound using the RELEASE(DEALLOCATE), this behavior took effect.

Figure 8-4 shows the Class 1 and Class 2 CPU per transaction. The results report 37% Total CPU reduction with PKGREL = COMMIT versus PKGREL = BNDOPT with AutoCommit ON.

Important: Packages bound with the RELEASE(DEALLOCATE) option in combination with HP DBATs and PKGREL = BNDOPT showed, for our particular test scenario, a 37% CPU savings for an application working with AutoCommit ON.

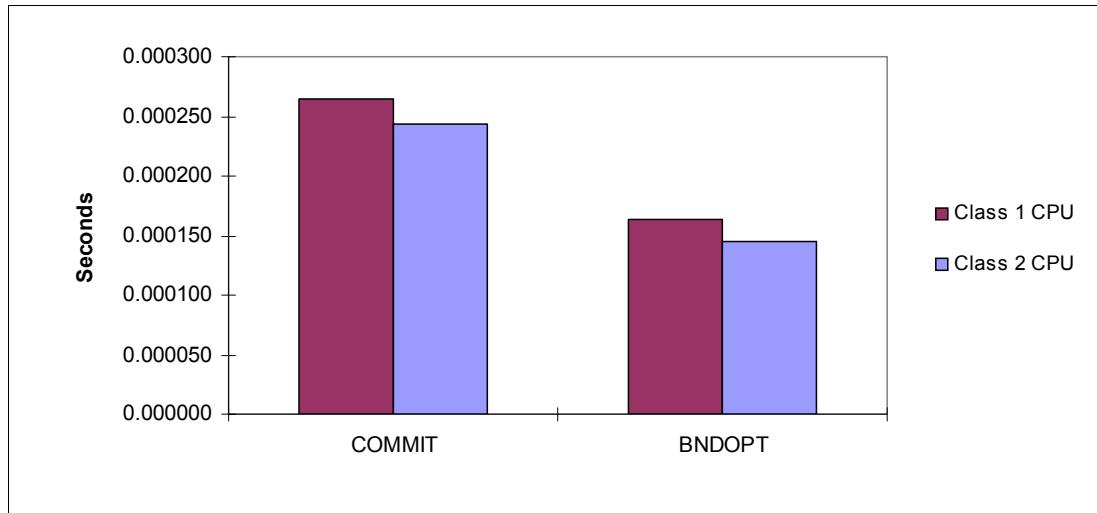


Figure 8-4 The DB2 10 distributed application: RELEASE(COMMIT) versus RELEASE(DEALLOCATE)

We conducted a series of tests with the objective of comparing DB2 9 and DB2 10 performance when working with RELEASE(COMMIT) and RELEASE(DEALLOCATE) for the following distributed workloads:

- ▶ SQCL: SQL ODBC / CLI (dynamic)
- ▶ SPCB: Stored Procedures in COBOL (static)
- ▶ JCC T4 Driver (DDF):
 - JDBC: Dynamic SQL
 - SQLJ: Static SQL
 - SPSJ: Stored procedures in SQLJ with static SQL
 - SPNS: Stored procedures in native SQL static

Total transaction CPU for SQCL, JDBC, SQLJ, and SPNS workloads is the sum of system services address space, database services address space, IRLM, and DDF address space CPU per commit as reported in OMEGAMON PE statistics report address space CPU section. Total transaction CPU for stored procedure workloads such as SPCB and SPCJ is the statistics total address space CPU per commit plus accounting class 1 STORED PRC CPU in the OMEGAMON PE report.

Table 8-1 shows the observed results, CPU time is reported in microseconds.

Table 8-1 RELEASE(COMMIT) versus RELEASE(DEALLOCATE) for distributed applications

Total CPU transaction (microsec.)	DB2 9	DB2 10 PKREL(COMMIT)	Delta %	DB2 10 PKREL(BNDOPT)	Delta %
SQCL	2114	1997	-5.5	1918	-9.3
SPCB	1221	1124	-7.9	1056	-13.5
JDBC	2152	2017	-6.3	1855	-13.8
SQLJ	1899	1761	-11.9	1689	-16.6
SPSJ	1768	1642	-6.7	1550	-11.9
SPNS	1472	1304	-11.4	1180	-19.8

High performance DBATs in combination with RELEASE(DEALLOCATE) for distributed application can improve performance and reduce elapsed time. Results can vary and the benefits are more pronounced for short transactions.

8.2 Limited block fetch extended to the JCC Type 2 drivers

Over the past several DB2 versions, DDF processing with DBM1 was optimized and zIIP redirection significantly reduced chargeable CP consumption. Other improvements included these:

- ▶ Limited block fetch
- ▶ LOB progressive streaming
- ▶ Implicit CLOSE

These improvements were not available to *local* Java and ODBC applications that did not always perform faster compared to the same application called remotely. These improvement to remote Java applications were described in the *DB2 9 for z/OS Performance Topics*, SG24-7473 and the *DB2 Version 9.1 for z/OS Application Programming and SQL Guide*, SC18-9841.

With DB2 10, many of these improvements are implemented for *local Java applications* using ODBC or JDBC. You can expect significant performance improvement for applications with the following queries:

- ▶ Queries that return more than 1 row
- ▶ Queries that return LOBs

Limited block fetch (LBF) support has been extended to the JCC Type 2 drivers on z/OS. This technology, already available in the JCC T4 and the distributed ODBC/CLI drivers, can provide dramatic improvements for applications involving large result set transfers; IBM observed more than 160% improvements in elapsed time and more than 170% improvements in CPU time in applications getting the advantages of this enhancement. This change leverages the drivers' functionalities and removes an inhibiting factor to the deployment of the T2 drivers for z based Java applications. The JCC Type 2 driver gets installed or updated automatically when DB2 10 is installed.

Figure 8-5 depicts the flow of calls from a Java application running on z/OS to DB2 9 on the same LPAR using the JDBC Type 2 driver.

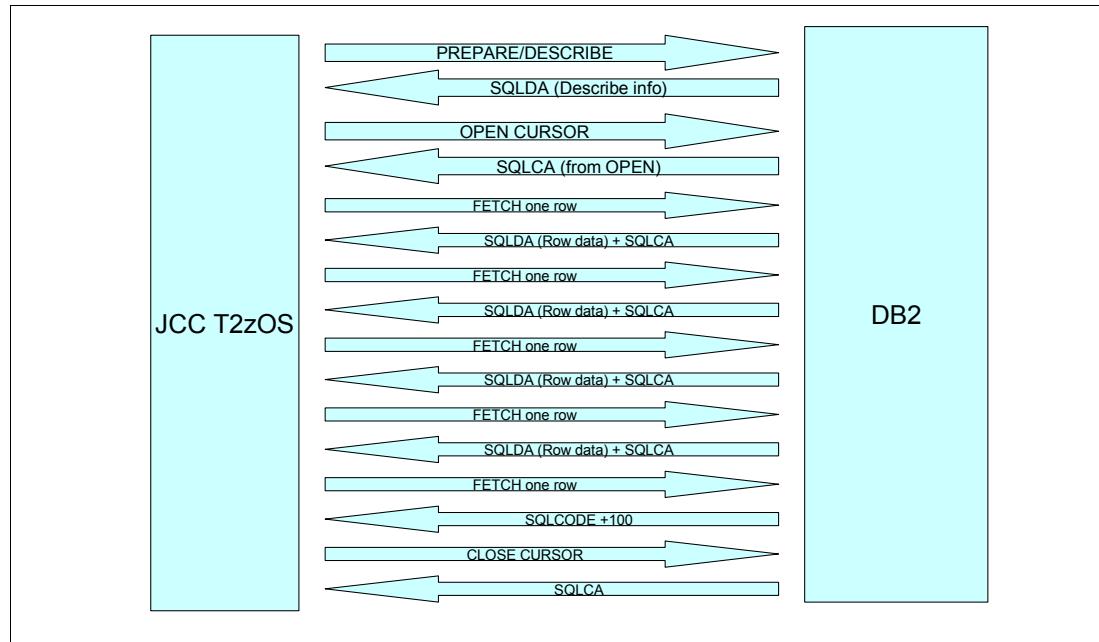


Figure 8-5 JDBC T2 driver in DB2 9 for z/OS

In this case, the Java application opens a connection to DB2, prepares an SQL statement for execution, and executes the SQL statement. Assuming that the SQL statement qualifies 100 rows, the application then issues a FETCH statement to get each row. Each FETCH results in a call to DB2 to return the data. After the application has processed the row, it issues the next FETCH statement, which means for this example, 100 calls. After all the rows have been fetched and processed by the application, it issues a CLOSE statement, which is another call. This process is irrespective whether the application runs in a application server such as IBM WebSphere® Application Server or in a stand-alone Java application if executed on z and exploiting the JCC Type 2 driver.

With limited block fetch, the DB2 for z/OS server attempts to fit as many rows as possible in a query block. Data can also be pre-fetched when the cursor is opened without needing to wait for an explicit fetch request from the requester. The immediate impact for the applications is a possible, in some cases even dramatic, reduction in the number of communication messages causing a drop in total elapsed time and CPU time.

Figure 8-6 depicts the sequence of calls from a Java application running on z/OS to DB2 on z/OS on the same LPAR using type 2 connectivity but this time against a DB2 10 subsystem.

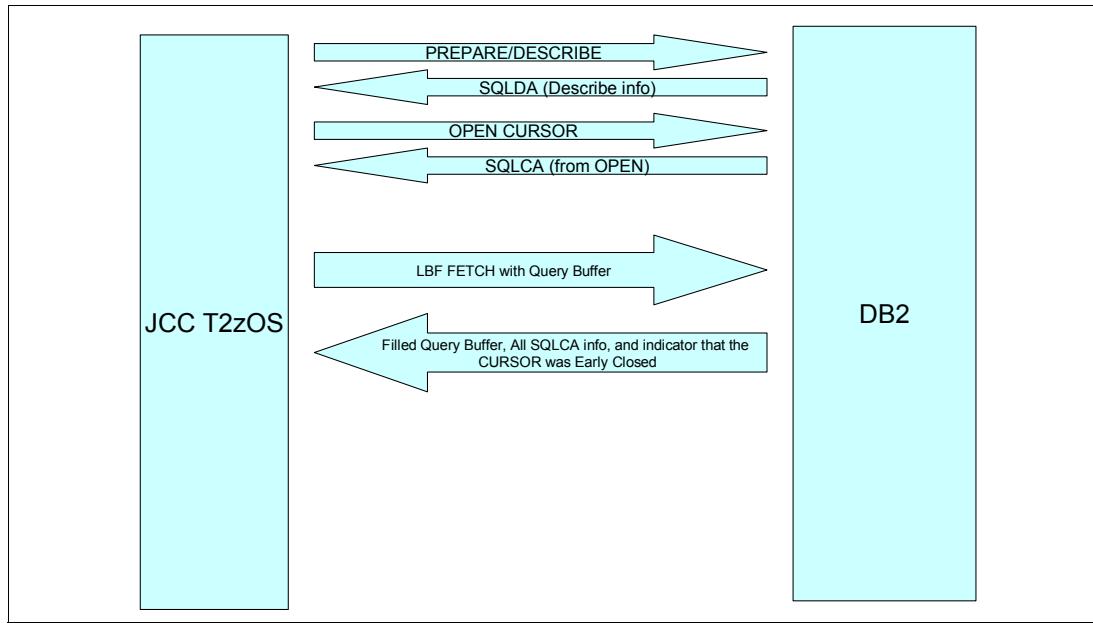


Figure 8-6 JDBC T2 driver in DB2 10 for z/OS

Because the JDBC type 2 driver is enhanced to provide limited block fetch capability, the driver returns as many rows as possible fitting in a buffer from a single FETCH call. This improvement is enabled by default, available in DB2 10 CM mode and there is no configuration required. It is not supported in JDBC/SQLJ stored procedures.

A series of tests were conducted in order to measure the performance benefits. The test environment involved DB2 10 for z/OS, z/OS 1.12 and the JCC drivers 3.61.84 and 4.11.86 in a system z z10 system.

8.2.1 Large result set

In this test, 3000 rows were fetched with and without limited block fetch support enabled. Figure 8-7 show a graphical representation of the observed results.

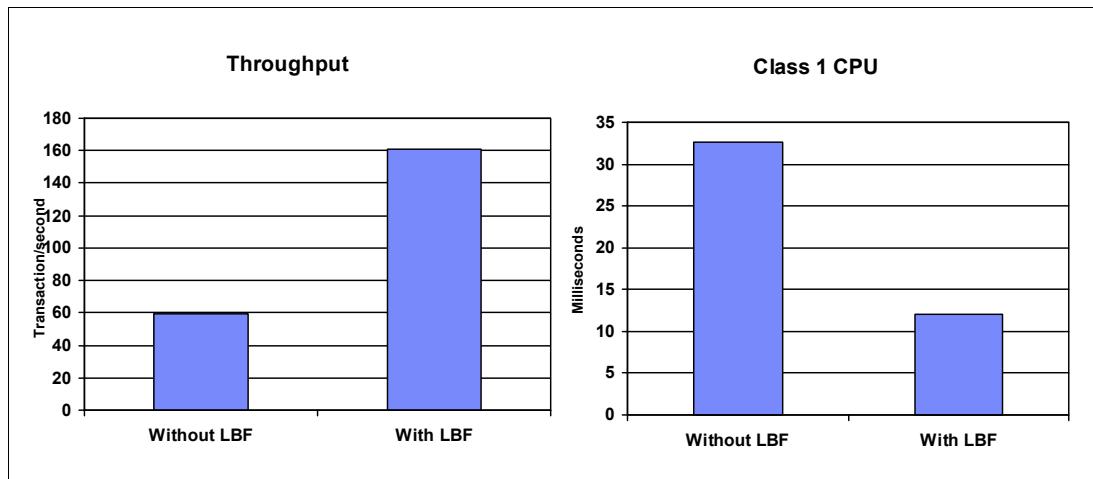


Figure 8-7 Limited block fetch: Large result set

These measures show the following results:

- ▶ Limited block fetch increases throughput by 169%.
- ▶ Limited block fetch reduces class 1 CPU time by 63%.

Example 8-8 shows a portion of the OMEGAMON PE accounting report for this test. This section shows that the number of FETCHs is lower than the number of ROWS, and this is an indication that limited block fetch was used for the data transfer during this test.

Example 8-8 Observing in the DB2 accounting if limited block fetch is active

SQL DML	AVERAGE
SELECT	0.00
INSERT	0.00
ROWS	0.00
UPDATE	0.00
ROWS	0.00
MERGE	0.00
DELETE	0.00
ROWS	0.00
DESCRIBE	1.00
DESC.TBL	0.00
PREPARE	1.00
OPEN	1.00
FETCH	53.00
ROWS	3000.00
CLOSE	0.00
DML-ALL	56.00

The number of rows returned per call depends on the buffer size, which is controlled by the queryDataSize property. queryDataSize specifies a hint that is used to control the amount of query data, in bytes, that is returned from the data source on each fetch operation. This value can be used to optimize the application by controlling the number of trips to the data source that are required to retrieve data.

A block of rows is returned to the Java application and all the rows that fit the buffer size are available in the JVM. The default is 32 KB; the use of a larger value for queryDataSize can result in less network traffic, which can result in better performance. For example, if the result set size is 50 KB, and the value of queryDataSize is 32 KB, two trips to the database server are required to retrieve the result set. However, if queryDataSize is set to 64 KB, only one trip to the data source is required to retrieve the result set.

The values that you can use for queryDataSize depends on the target database. Table 8-2 shows the values that you can use when the target is DB2 for z/OS. This table shows that the maximum value for queryDataSize has been increased 4 times when comparing DB2 9 and DB2 10.

Table 8-2 Default, minimum, and maximum values of queryDataSize

Data source	Version	Default	Minimum	Maximum
DB2 for z/OS	Version 8	32767	32767	32767
DB2 for z/OS	Version 9	32767	32767	65535
DB2 for z/OS	Version 10	32767	32767	262143

Appropriate tuning of the DataSource property queryDataSize can improve performance by reducing the number of messages required between DB2 10 and a Java application when using JDBC T2 driver and running on z/OS. This property also applies to the JDBC Type 4 driver. Consider using a queryDataSize value bigger than 32 KB for large result sets if the utilization of a bigger buffer reduces the number of messages between DB2 and the application.

8.2.2 Single row result set

The JDBC Type 2 driver is enhanced to implement early close of a cursor after all the rows are returned. Instead of a separate call to close the cursor from the application, the driver closes the cursor in DB2 implicitly.

For a single row result set, this change reflects in a single FETCH instead of 2 in an OMEGAMON PE Accounting report, as illustrated in Example 8-9.

Example 8-9 Accounting for single row result set with and without limited block fetch

NO LBF		WITH LBF	
SQL	DML	SQL	DML
AVERAGE		AVERAGE	
SELECT	0.00	SELECT	0.00
INSERT	0.00	INSERT	0.00
ROWS	0.00	ROWS	0.00
UPDATE	0.00	UPDATE	0.00
ROWS	0.00	ROWS	0.00
MERGE	0.00	MERGE	0.00
DELETE	0.00	DELETE	0.00
ROWS	0.00	ROWS	0.00
DESCRIBE	1.00	DESCRIBE	1.00
DESC.TBL	0.00	DESC.TBL	0.00
PREPARE	1.00	PREPARE	1.00
OPEN	1.00	OPEN	1.00
FETCH	2.00	FETCH	1.00
ROWS	1.00	ROWS	1.00
CLOSE	1.00	CLOSE	0.00
DML-ALL	7.00	DML-ALL	5.00

There is, however, a slight performance degradation for this type of transaction when limited block fetch is enabled: even if the DB2 Class 2 CPU time is less due to fewer FETCHes, there is a small increment in the CPU used by DDF for transferring the block of data. Figure 8-8 shows this overhead.

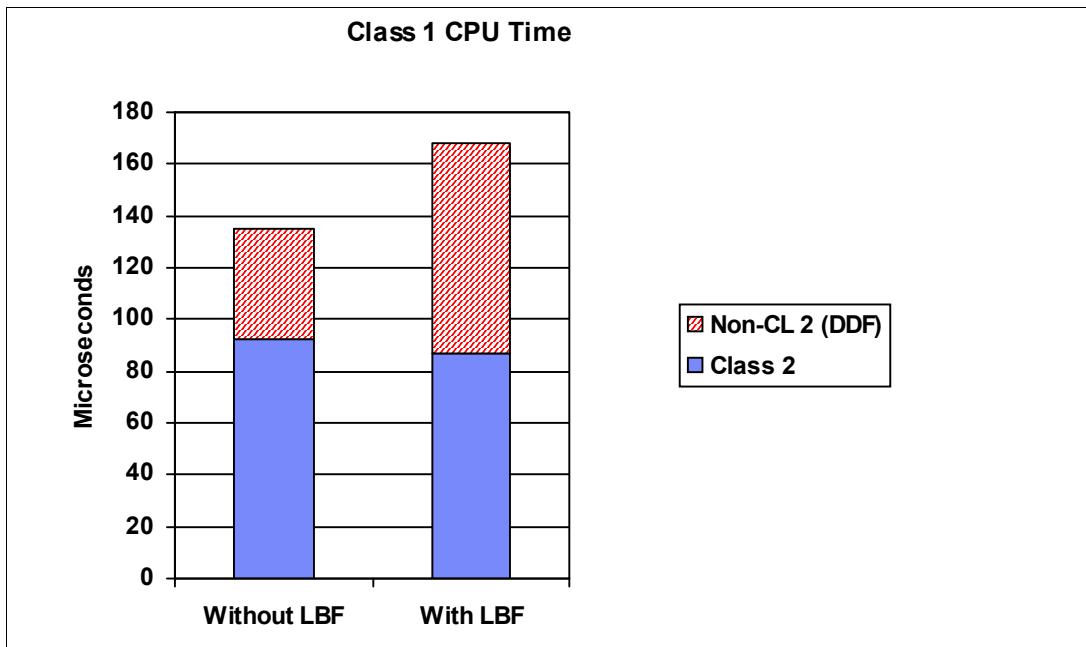


Figure 8-8 Limited block fetch overhead for single row result set

For single row result sets when limited block fetch is enabled, we observed these impacts:

- ▶ No change in throughput
- ▶ 24.4% increase in Class 1 CPU time due to more activity in DDF
- ▶ 5% decrease in Class 2 CPU time due to less FETCHs

8.2.3 IRWW workload

We also used the IRWW distributed version of the workload in order to verify the impact of limited block fetch. IRWW is described in 5.1.2, “IRWW workload” on page 127. There are seven transactions. Each transaction consists of one to many SQL statements, each performing a distinct business function in a predefined mix.

Figure 8-9 shows a graphical representation of the observations.

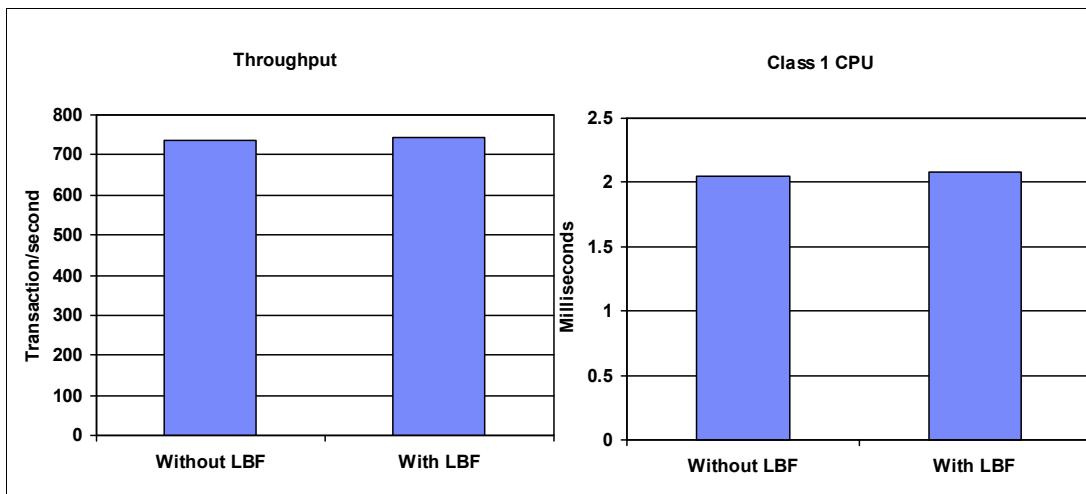


Figure 8-9 Limited block fetch: IRWW OLTP workload

IRWW is an OLTP workload and the effect of activating limited block fetch is inline with the expectations for a single result set kind of workload. There is no throughput degradation and a 3% increase in Class 1 CPU time.

8.2.4 Limited block fetch summary

The advantages of limited block fetch have been extended to z/OS JCC Type 2 driver. Controlled tests exhibit dramatic performance improvements, for example:

- ▶ Limited block fetch increases throughput by 169%.
- ▶ Limited block fetch reduces class 1 CPU time by 63%.

Results can vary depending on the workload, but this enhancement certainly makes the z/OS JCC Type 2 Driver an option to be considered.

Make sure to limit the result set to the actual number of rows needed. This can be done through the `FETCH FIRST x ROWS ONLY` SQL clause. Failing to do so can cause situations, with limited block fetch enabled, where the entire result set is retrieved even when only a single row or very few rows are requested (fetched) by the application. This might cause a performance degradation.

8.3 Return to client result sets

Prior to DB2 10, a stored procedure can only return result sets to the immediate caller. If the stored procedure is in a chain of nested calls, the result sets must be materialized at each intermediate nesting level, typically through a declared global temporary table (DGTT).

DB2 10 introduces *return to client result set support*. With this enhancement, a result set can be returned from a stored procedure at any nesting level directly to the client calling application. No materialization through DGTTs is required. The new syntax on the `DECLARE CURSOR` statement is as follows:

```
WITH RETURN TO CLIENT
```

Result sets that are defined `WITH RETURN TO CLIENT` are not visible to any stored procedures at the intermediate levels of nesting. They are only visible to the client that issued the initial `CALL` statement. This feature is not supported for stored procedures called from triggers or functions, either directly or indirectly.

8.3.1 Test scenarios

We ran four tests involving nested stored procedures in order to measure the performance of this enhancement. The four test scenarios are outlined next.

Test 1 (DB2 9 DGTT): Base case

- ▶ Client application calls stored procedure SP1
- ▶ Stored procedure SP1 declares a global temporary table and calls stored procedure SP2
- ▶ Stored procedure SP2 declares a cursor WITH RETURN TO CALLER to pass a result set
- ▶ Stored procedure SP1 fetches from the result set and inserts into the declared temporary table
- ▶ Client application reads from the declared temporary table

Test 2 (DB2 10 DGTT): Same test as test 1 but run on DB2 10 NFM

- ▶ Client application calls stored procedure SP1
- ▶ Stored procedure SP1 declares a global temporary table and calls stored procedure SP2
- ▶ Stored procedure SP2 declares a cursor WITH RETURN TO CALLER to pass a result set
- ▶ Stored procedure SP1 fetches from the result set and inserts into the declared temporary table
- ▶ Client application reads from the declared temporary table

Test 3 (DB2 10 CGTT): Using a created temporary table NFM

- ▶ Client application calls stored procedure SP1
- ▶ Stored procedure SP1 calls stored procedure SP2
- ▶ Stored procedure SP2 declares a cursor WITH RETURN TO CALLER to pass a result set
- ▶ Stored procedure SP1 fetches from the result set and inserts into a created temporary table
- ▶ Client application reads from the created temporary table

Test 4 (DB2 10 RTC): Returning results to client NFM

- ▶ Client application calls stored procedure SP1
- ▶ Stored procedure SP1 calls stored procedure SP2
- ▶ Stored procedure SP2 declares a cursor WITH RETURN TO CLIENT to pass a result set
- ▶ Stored procedure SP1 returns control to the client application
- ▶ Client application fetches from the result set

8.3.2 Test results

We ran each of the four tests twice: once with a single row result set; and once with a 500 row result set. The results of all eight performance measurements are in the following charts. The four tests are identified as follows in the charts. All measurements are in milliseconds.

The class 1 CPU times for the four tests with a single row result set are shown in Figure 8-10.

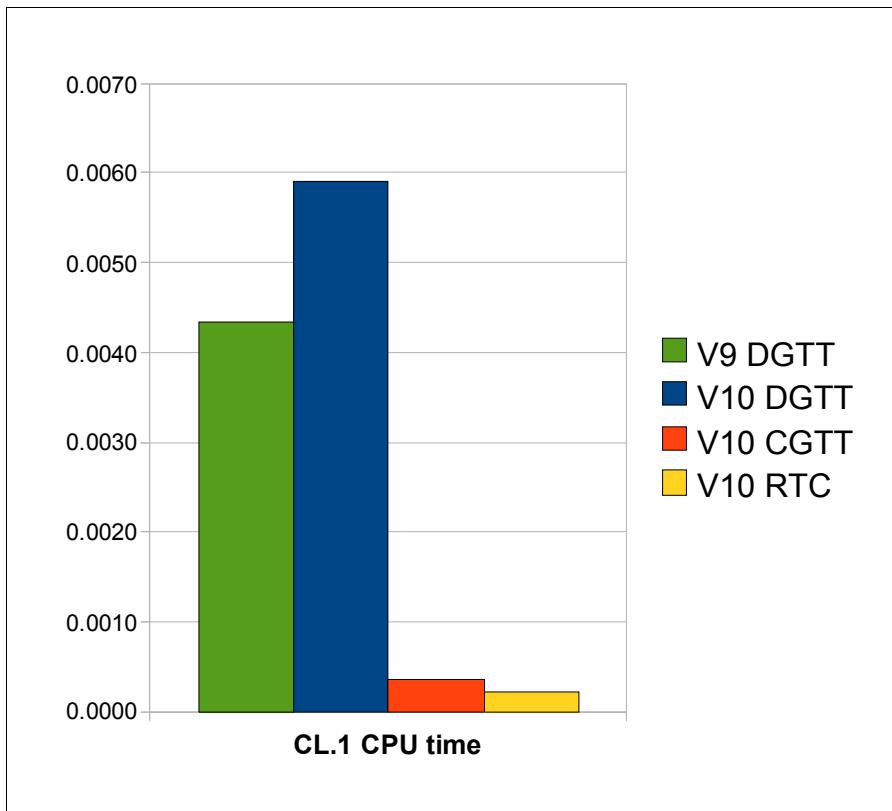


Figure 8-10 Class 1 CPU measurements for WITH RETURN TO CLIENT - Single row result set

For the single row result set tests, DB2 10 incurs a class 1 CPU increase of 36% over DB2 9 when using declared global temporary tables (DGTTs), represented by an increase from 4.35 milliseconds to 5.92 milliseconds of class 1 CPU time. When we use a created global temporary table (CGTT) instead of a DGTT, we see a 94% reduction in class 1 CPU time over the same test in DB2 10 with a DGTT, with the DGTT test consuming 5.92 milliseconds of class 1 CPU time and the CGTT test consuming 0.36 milliseconds of class 1 CPU time.

The test using WITH RETURN TO CLIENT consumes 0.21 milliseconds of class 1 CPU time. In new function mode, the use of WITH RETURN TO CLIENT provides the following percentage savings in class 1 CPU time over the other three tests:

- ▶ 95% savings over DB2 9 using a DGTT
- ▶ 96% savings over DB2 10 new-function mode using a DGTT
- ▶ 42% savings over DB2 10 new-function mode using a CGTT

The class 2 CPU savings for the single row result set are similar to the class 1 CPU savings. The class 2 CPU times for the four tests with a single row result set are shown in Figure 8-11.

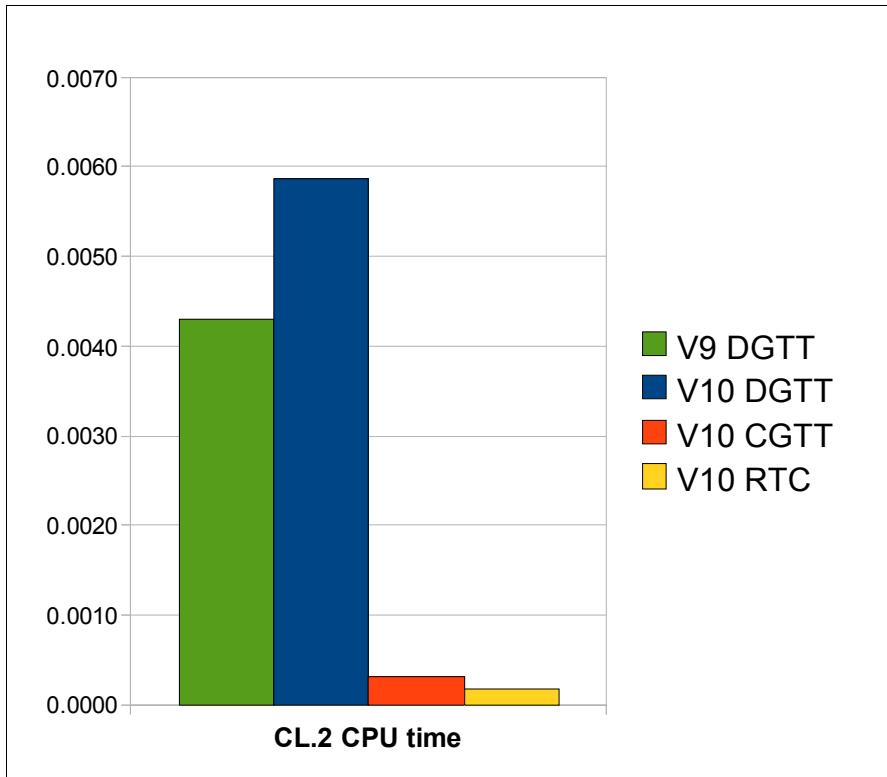


Figure 8-11 Class 2 CPU measurements for WITH RETURN TO CLIENT - Single row result set

The test using WITH RETURN TO CLIENT consumes 0.17 milliseconds of class 2 CPU time, compared to 4.29 milliseconds for DB2 9 with a DGTT, 5.86 milliseconds for DB2 10 with a DGTT and 0.32 seconds for DB2 10 with a CGTT.

In new function mode, the use of WITH RETURN TO CLIENT provides the following percentage savings in class 2 CPU time over the other three tests:

- ▶ 96% savings over DB2 9 using a DGTT
- ▶ 97% savings over DB2 10 conversion mode using a DGTT
- ▶ 47% savings over DB2 10 conversion mode using a CGTT

The tests for the single row result set show significant savings when using WITH RETURN TO CLIENT because of the savings for not needing to use a temporary table, whether a declared one or a created one. However, the bigger savings come when we have a larger result set and, therefore, a larger temporary table that can be eliminated.

We ran the same four tests using a result set of 500 rows instead of a single row. The class 1 CPU times for the four tests with a 500 row result set are shown in Figure 8-12.

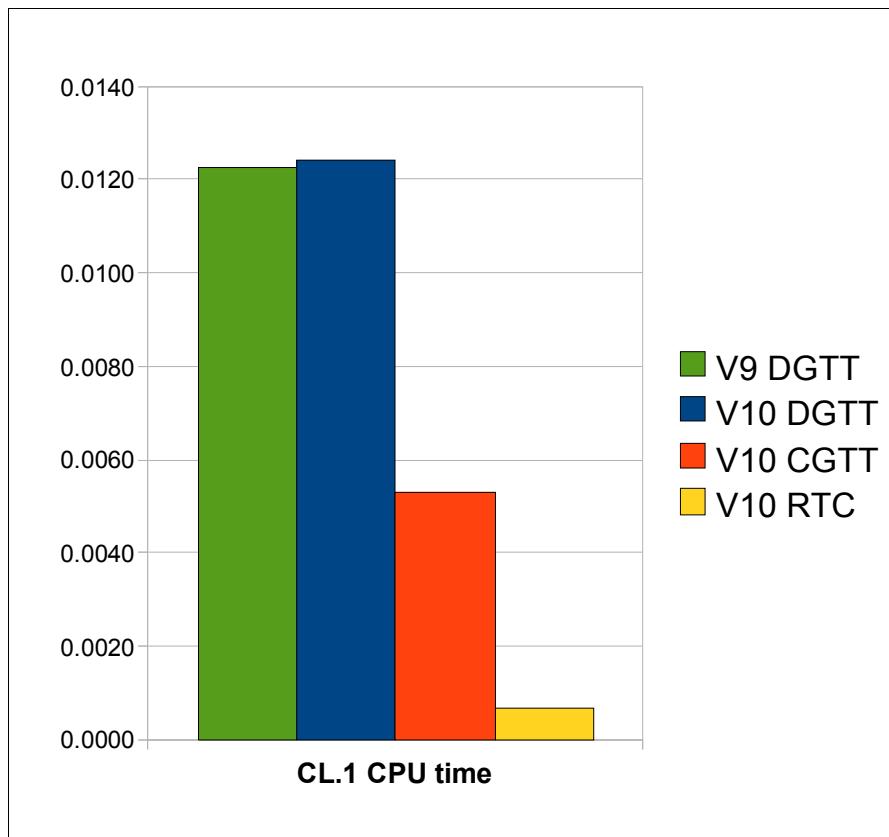


Figure 8-12 Class 1 CPU measurements for WITH RETURN TO CLIENT - 500 row result set

For the 500 row result set tests, the test using WITH RETURN TO CLIENT consumes 0.67 milliseconds of class 1 CPU time, compared to 12.28 milliseconds for DB2 9 with a DGTT, 12.41 milliseconds for DB2 10 with a DGTT and 5.30 seconds for DB2 10 with a CGTT.

In new function mode, the use of WITH RETURN TO CLIENT provides the following percentage savings in class 1 CPU time over the other three tests:

- ▶ 94% savings over DB2 9 using a DGTT
- ▶ 95% savings over DB2 10 conversion mode using a DGTT
- ▶ 87% savings over DB2 10 conversion mode using a CGTT

The class 2 CPU savings for the 500 row result set are similar to the class 1 CPU savings. The class 2 CPU times for the four tests with a 500 row result set are shown in Figure 8-13.

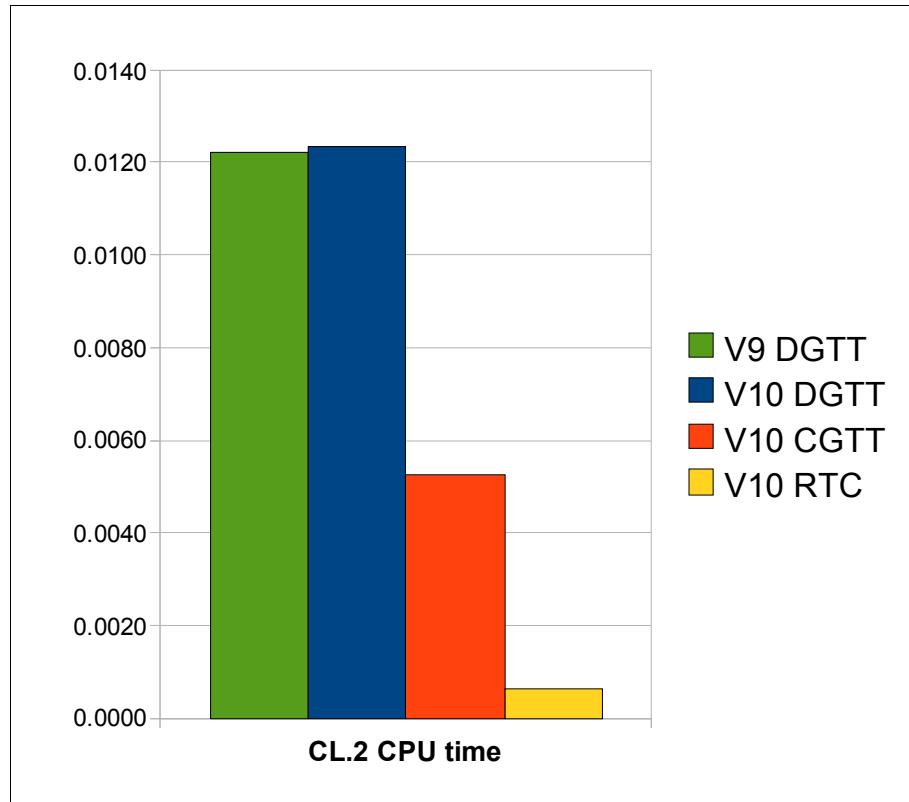


Figure 8-13 Class 2 CPU measurements for WITH RETURN TO CLIENT - 500 row result set

For the 500 row result set tests, the test using WITH RETURN TO CLIENT consumes 0.63 milliseconds of class 2 CPU time, compared to 12.21 milliseconds for DB2 9 with a DGTT, 12.34 milliseconds for DB2 10 with a DGTT and 5.25 seconds for DB2 10 with a CGTT.

In new function mode, the use of WITH RETURN TO CLIENT provides the following percentage savings in class 2 CPU time over the other three tests:

- ▶ 95% savings over DB2 9 using a DGTT
- ▶ 95% savings over DB2 10 conversion mode using a DGTT
- ▶ 88% savings over DB2 10 conversion mode using a CGTT

The processing of declared global temporary tables (DGTTs) is more expensive in DB2 10 due to the catalog restructure (now indexed versus direct links). Users might see a decrease in performance in DB2 10 when using DGTTs. In cases where this is due to the materialization of result sets within stored procedures, using the new return to client result set support can significantly improve performance.

The size of the result set also impacts the percentage of savings you experience when using WITH RETURN TO CLIENT instead of a temporary table. The greater the size of the result set, the greater the size of the temporary table that does not need to be passed through each nesting level.

The return to client result sets enhancement is available in new-function mode.

8.4 Enhanced support for native SQL procedures

DB2 9 for z/OS introduced native SQL procedures. The SQL procedural language (SQL PL) is the language used to develop native SQL procedures. DB2 10 provides a number of functional enhancements as well as the following performance enhancements:

- ▶ The SQL PL assignment statement, SET, is extended to allow you to set multiple values with a single SET statement. This is similar to the existing support for SET:host-variable statement.
- ▶ The path length for evaluating IF statements has been reduced.
- ▶ CPU has been reduced for some SET statements that make reference to built-in functions. This enhancement does not include references to user defined functions (UDFs) or references to XML, user defined data types (UDTs) or special registers in the statement. For example, SET INT_COLA = INT(CHAR_COLB) will benefit from these CPU savings.

Performance results for an OLTP workload using SQL PL show a 20% reduction in CPU and a 5% improvement in response time, running in DB2 10 in CM. The SQL procedures need to be regenerated first.

You can gain some significant performance enhancements for native SQL procedures without changing your code. You can gain additional savings by performing multiple assignments within a single SET statement. Example 8-10 shows SET assignments.

Example 8-10 Chaining SET statements

Non-chained	Chained
-----	-----
SET x=1;	SET x=1, y=2, z=3;
SET y=2;	
SET z=3;	

The performance enhancements for native SQL procedures are available in conversion mode; however you need to recreate or regenerate your SQL PL procedures to take full advantage of all of these performance enhancements.

8.5 Extended correlation token

Historically, correlation of work between a DB2 for z/OS and any associated remote partners has been through the logical unit of work identifier (LUWID). This concept was introduced with the initial SNA distributed support and continued to be used when TCP/IP support was introduced, because an IPv4 (32-bit) address can still be represented successfully (8 or 4-byte character form) in the LUWID.

With the introduction of IPv6 support in DB2 9, an IPv6 (128-bit) address can no longer be represented in an LUWID, and the concept of an *extended correlation token* was introduced. This extended correlation token represented the entire IPv6 address. DB2 9 provides this extended correlation token only in **DISPLAY THREAD** command reports and trace records.

DB2 10 provides the extended correlation token in more messages, making it much easier to correlate message-related failures to the remote client application that is involved in the failure.

Example 8-11 shows a DB2 9 example of a distributed timeout.

Example 8-11 DB2 9 for z/OS report of a distributed application being TIMED OUT

```
18.53.44 STC08407 DSNT376I -DB9A PLAN=DISTSERV WITH 824
824          CORRELATION-ID=javaw.exe
824          CONNECTION-ID=SERVER
824          LUW-ID=G91E1CC0.E2FB.110222235238=210
824          THREAD-INFO=DB2R1:CRIS:db2r1:javaw.exe
824          IS TIMED OUT. ONE HOLDER OF THE RESOURCE IS PLAN=DSNESPCS
824          WITH
824          CORRELATION-ID=DB2R1
824          CONNECTION-ID=TSO
824          LUW-ID=USIBMSC.SCPDB9A.C75F7D5848B0=201
824          THREAD-INFO=DB2R1:*:*:*
824          ON MEMBER DB9A
```

Example 8-12 shows a similar process as reported in DB2 10.

Example 8-12 DB2 10 for z/OS report showing extended correlation token information

```
19.01.41 STC12396 DSNT376I -DBOA PLAN=DISTSERV WITH 831
831          CORRELATION-ID=javaw.exe
831          CONNECTION-ID=SERVER
831          LUW-ID=G91E1CC0.E339.110223000109=5209
831          THREAD-INFO=DB2R1:CRIS:db2r1:javaw.exe:DYNAMIC:0:*:<9.30.28.192.58169
831          .110223000109>
831          IS TIMED OUT. ONE HOLDER OF THE RESOURCE IS PLAN=DSNESPCS
831          WITH
831          CORRELATION-ID=DB2R1
831          CONNECTION-ID=TSO
831          LUW-ID=USIBMSC.SCPDBOA.C75F8130F43C=5206
831          THREAD-INFO=DB2R1:*:*:*:DYNAMIC:1:*
831          ON MEMBER DBOA
```

For the specific case of the DB2 message DSNT376I, the THREAD-INFO details are presented in a colon-delimited list that contains the following segments:

- ▶ The primary authorization ID that is associated with the thread. In many distributed configurations, the primary authorization ID that is used with DB2 is not necessarily the user's ID.
- ▶ The name of the user's workstation.
- ▶ The ID of the user.
- ▶ The name of the application.
- ▶ The statement type for the currently executing statement: dynamic or static.
- ▶ The statement identifier for the currently executing statement, if available. The statement identifier can be used to identify the particular SQL statement. For static statements, the statement identifier correlates to the STMT_ID column in the SYSIBM.SYSPACKSTMT table. For dynamic statements, the statement identifier correlates to the STMT_ID column in the DSN_STATEMENT_CACHE_TABLE table.
- ▶ The name of the role that is associated with the thread.
- ▶ The correlation token that can be used to correlate work at the remote system with work performed at the DB2 subsystem. The correlation token, if available, is enclosed in '<' and '>' characters, and contains three components, separated by periods:
 - A 3 to 39 character IP address.
 - A 1 to 8 character port address.
 - A 12 character unique identifier

An asterisk (*) in any segment indicates that the information is not available.

The extended information provided by DB2 10 for z/OS allows us, in cases such as this one, to quickly identify the workstation or server at the origin of the TIMED OUT statement.

8.6 Virtual and real storage with distributed IRWW workload

DB2 10 for z/OS provides a dramatic reduction of virtual private storage below the bar, moving 50-90% of the current storage above the bar by exploiting 64-bit virtual storage. This change allows as much as 10 times more concurrent active tasks in DB2. In order to investigate storage utilization in DB2 10 for distributed workloads, several measurements were done under these conditions:

- ▶ JDBC Type 4 IRWW workload
- ▶ Number of concurrent connections: 500, 1000, 1500.
- ▶ Tracking DB2 storage using IFCID 225 and RMF report

Figure 8-14 shows the storage used below the bar, in Megabytes, for distributed workloads comparing DB2 9 to DB2 10.

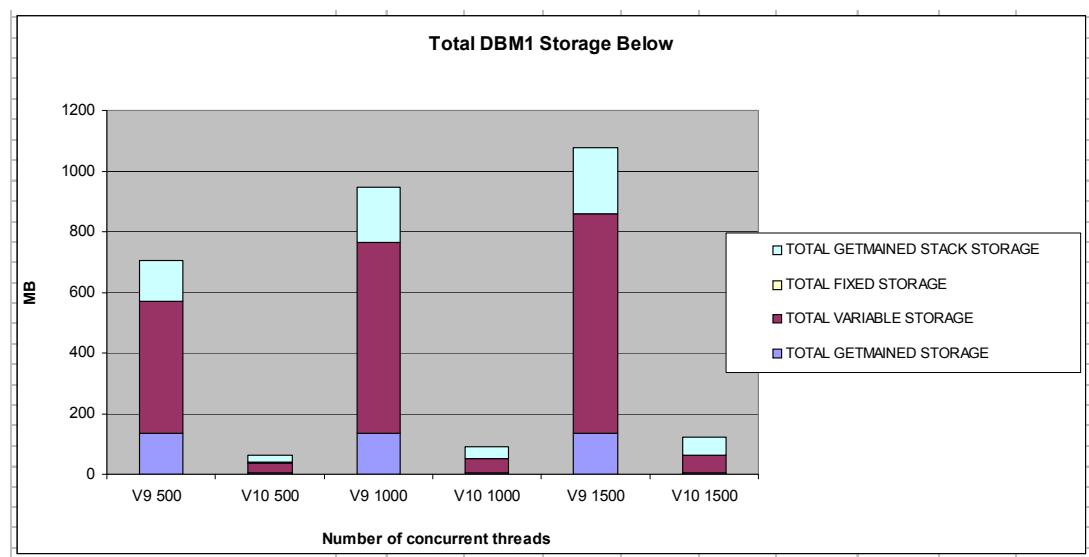


Figure 8-14 DBM1 storage below the bar, distributed workload 9 versus 10

Measurement results were as follows:

- ▶ The total DBM1 utilization below the 2 GB bar is reduced by 90%.
- ▶ Common storage per thread is reduced by 50%.

This dramatic reduction on storage requirements is the base of the DB2 10 scalability characteristics.

There is, however, an observed real storage demand increase ranging from 3% to 12%. This is illustrated in Figure 8-15 where storage values are measured in GB.

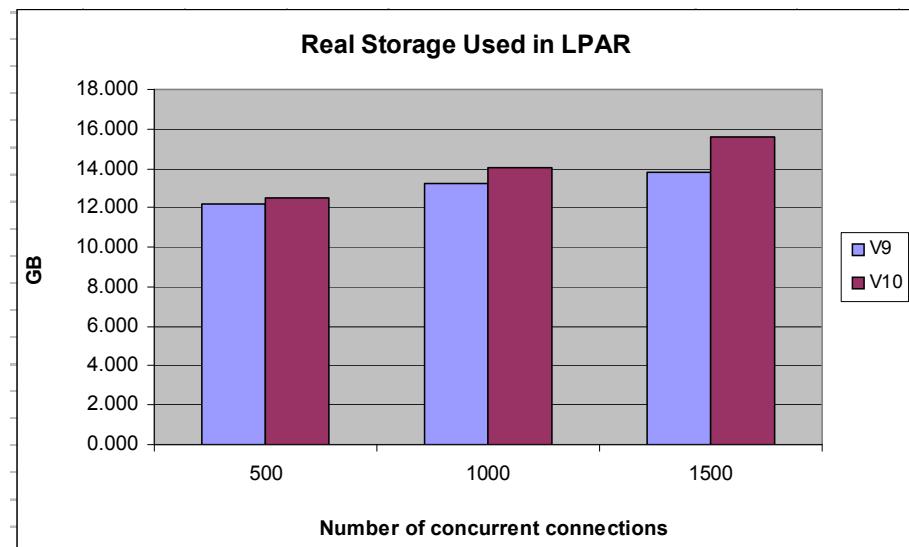


Figure 8-15 Distributed workload: DB2 9 versus DB2 10 total real storage utilization compared

In DB2 10 IFCID 225 supports virtual storage report on DIST address space. You can report on IFCID 225 using the RECTRACE report of OMEGAMON PE. Example 8-13 shows an example of OMEGAMON PE JCL showing the RECTRACE syntax.

Example 8-13 OMEGAMON PE RECTRACE report syntax

```
//PE      EXEC PGM=FPECMAN
//STEPLIB DD DISP=SHR,DSN=OMEGASYS.DBOA.BASE.RKANMOD
//INPUTDD DD   DISP=SHR,DSN=SMFDATA.DB2RECS.G5383V00
//JOBSUMDD DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//ACRPTDD DD SYSOUT=*
//UTTRCDD1 DD SYSOUT=*
//SYSIN    DD *
GLOBAL
      TIMEZONE (+ 05:00)
RECTRACE
      TRACE
      LEVEL(LONG)
      INCLUDE(SUBSYSTEM(DB0A))
      INCLUDE(PRIMAUTH(DB2R1))
EXEC
/*
```

Example 8-14 shows a portion of an OMEGAMON PE RECTRACE report with DIST address space utilization. Note that a large part of the report has been removed for clarity.

Example 8-14 Reporting DIST address space storage with IFCID 225

LOCATION: DBOA GROUP: N/P MEMBER: N/P SUBSYSTEM: DBOA DB2 VERSION: V10 OPRIMAUTH CONNECT ORIGAUTH CORRNNAME PLANNAME CORRNMBR	INSTANCE CONNTYPE RECORD TIME TCB CPU TIME	END_USER DESTNO ACE IFC WS_NAME TIME ID	TRANSACT DATA DESCRIPTION	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1) RECORD TRACE - SHORT PAGE: 1-9 REQUESTED FROM: NOT SPECIFIED TO: NOT SPECIFIED ACTUAL FROM: 03/10/11 13:02:00.00 PAGE DATE: 03/10/11
-----	-----	-----	-----	-----

N/P	N/P	C772654CC707	N/P	N/P	N/P	
N/P	N/P	'BLANK'	13:02:00.00566165	25794	1 225	STORAGE MGR
N/P			N/P		NETWORKID:	DBOA
					LUNAME:	DBOA
					LUWSEQ:	1
<hr/>						
ADDRESS SPACE SUMMARY - DIST						
EXTENDED REGION SIZE (MAX) : 1521483776 24-BIT LOW PRIVATE : 245760						
24-BIT HIGH PRIVATE : 262144 31-BIT EXTENDED LOW PRIVATE : 13570048						
31-BIT EXTENDED HIGH PRIVATE : 17080320 CURRENT HIGH ADDRESS 24-BIT PRIVATE REGION : X'00042000'						
CURRENT HIGH ADDRESS 31-BIT PRIVATE REGION : X'261F1000' 31-BIT RESERVED FOR MUST COMPLETE : 152148377						
31-BIT RESERVED FOR MVS : 26040960 STORAGE CUSHION WARNING TO CONTRACT : 152148377						
TOTAL 31-BIT GETMAINED STACK : 4124672 TOTAL 31-BIT STACK IN USE : 4124672						
TOTAL 31-BIT VARIABLE POOL : 761856 TOTAL 31-BIT FIXED POOL : 86016						
TOTAL 31-BIT GETMAINED : 45210 AMOUNT OF AVAILABLE 31-BIT : 1490829312						
TOTAL 64-BIT VARIABLE POOL : 1466368 TOTAL 64-BIT FIXED : 118784						
TOTAL 64-BIT GETMAINED : 0 TOTAL 64-BIT PRIVATE FOR STOR MANAG: 1400832						
REAL 4K FRAMES IN USE : 5019 AUXILIARY SLOTS IN USE : 0						
64-BIT REAL 4K FRAMES IN USE : 605 64-BIT 4K AUX SLOTS IN USE : 0						
HWM 64-BIT REAL 4K FRAMES IN USE : 605 HWM 64-BIT AUX SLOTS IN USE : 0						
<hr/>						

OMEGAMON PE 5.1 has been updated to report the DIST address space storage in the statistics trace as well. Example 8-15 shows a syntax sample. Storage information for DIST is listed in the LONG report but not in the SHORT version.

Example 8-15 OMEGAMON PE Statistics report syntax

```
//PE      EXEC PGM=FPECMAN
//STEPLIB DD DISP=SHR,DSN=OMEGASYS.DBOA.BASE.RKANMOD
//INPUTDD DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5383V00
//JOBSUMDD DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//ACRPTDD DD SYSOUT=*
//UTTRCDD1 DD SYSOUT=*
//SYSIN    DD *
STATISTICS
  REPORT
    LAYOUT(LONG)
  GLOBAL
    INCLUDE(SSID(DBOA))
EXEC
/*
```

Example 8-16 shows an extract of an OMEGAMON PE statistics report long showing DIST storage below the 2 GB bar and real and auxiliary storage utilization.

Example 8-16 Reporting DIST address space storage in OMEGAMON PE Statistics Report Long

LOCATION: DBOA	OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1)	PAGE: 1-14
GROUP: N/P	STATISTICS REPORT - LONG	REQUESTED FROM: NOT SPECIFIED
MEMBER: N/P		TO: NOT SPECIFIED
SUBSYSTEM: DBOA		INTERVAL FROM: 11-03-14 21:06:00.00
DB2 VERSION: V10	SCOPE: MEMBER	TO: 11-03-14 21:09:00.00
<hr/>		
---- HIGHLIGHTS ----		
INTERVAL START : 11-03-14 21:06:00.00	SAMPLING START: 11-03-14 21:06:00.00	TOTAL THREADS : 3.00
INTERVAL END : 11-03-14 21:09:00.00	SAMPLING END : 11-03-14 21:09:00.00	TOTAL COMMITS : 6.00
INTERVAL ELAPSED: 2:59.999803	OUTAGE ELAPSED: 0.000000	DATA SHARING MEMBER: N/A
<hr/>		
DIST AND MVS STORAGE BELOW 2 GB		
QUANTITY		
TOTAL DIST STORAGE BELOW 2 GB (MB)	4.62	FIXED STORAGE (MB) 0.11
TOTAL GETMAINED STORAGE (MB)	0.00	GETMAINED STORAGE (MB) 0.00
TOTAL VARIABLE STORAGE (MB)	0.55	VARIABLE STORAGE (MB) 0.14
NUMBER OF ACTIVE CONNECTIONS	0.00	STORAGE MANAGER CONTROL BLOCKS (MB) 1.34
NUMBER OF INACTIVE CONNECTIONS	1.00	
TOTAL FIXED STORAGE (MB)	0.08	

TOTAL GETMAINED STACK STORAGE	(MB)	3.98		
TOTAL STACK STORAGE IN USE	(MB)	0.73		
STORAGE CUSHION	(MB)	315.03		
24 BIT LOW PRIVATE	(MB)	0.23		
24 BIT HIGH PRIVATE	(MB)	0.25		
24 BIT PRIVATE CURRENT HIGH ADDRESS		0000000000042000		
31 BIT EXTENDED LOW PRIVATE	(MB)	12.94		
31 BIT EXTENDED HIGH PRIVATE	(MB)	16.09		
31 BIT PRIVATE CURRENT HIGH ADDRESS		00000000261F1000		
EXTENDED REGION SIZE (MAX)	(MB)	1451.00		
REAL AND AUXILIARY STORAGE FOR DBM1	QUANTITY		REAL AND AUXILIARY STORAGE FOR DIST	QUANTITY
REAL STORAGE IN USE	(MB)	294.22	REAL STORAGE IN USE	(MB)
31 BIT IN USE	(MB)	84.73	31 BIT IN USE	(MB)
64 BIT IN USE	(MB)	209.49	64 BIT IN USE	(MB)
HWM 64 BIT REAL STORAGE IN USE	(MB)	209.49	HWM 64 BIT REAL STORAGE IN USE	(MB)
AUXILIARY STORAGE IN USE	(MB)	0.00	AUXILIARY STORAGE IN USE	(MB)
31 BIT IN USE	(MB)	0.00	31 BIT IN USE	(MB)
64 BIT IN USE	(MB)	0.00	64 BIT IN USE	(MB)
HWM 64 BIT AUX STORAGE IN USE	(MB)	0.00	HWM 64 BIT AUX STORAGE IN USE	(MB)
COMMON STORAGE BELOW AND ABOVE 2 GB	QUANTITY		MVS LPAR SHARED STORAGE ABOVE 2 GB	QUANTITY

8.7 LOBs and XML materialization avoidance

DB2 9 introduced functionality that eliminates the need for client application to read the entire LOB to get its total length prior to sending the LOB to the server. However, DB2 for z/OS still had to materialize the entire LOB in memory to get the length of the entire LOB before inserting it into the database.

DB2 10 for z/OS adds a performance improvement by extending the support for LOB and XML streaming at the host, avoiding LOB and XML materialization in most situations.

Figure 8-16 shows the differences between DB2 9 and DB2 10. The DDF server no longer needs to wait for the entire LOB or XML to be received and fully materialized before it can pass the LOB or XML to the data manager.

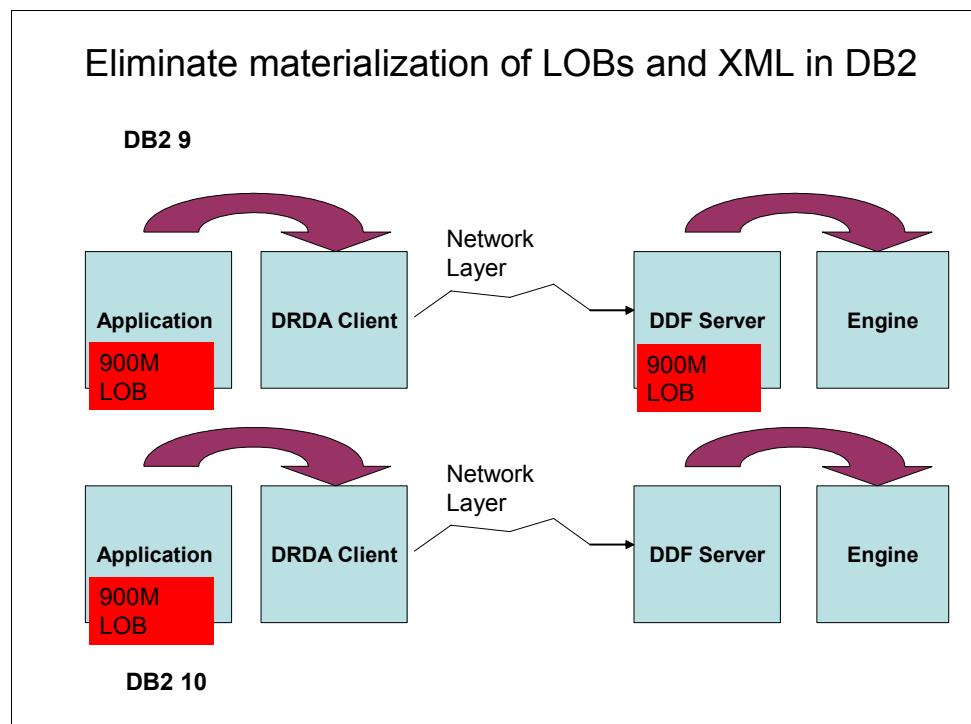


Figure 8-16 Streaming LOBs and XML

DB2 materializes up to 2 MB for LOBs and 32 KB for XML before passing the data to the database manager. The storage allocated for this LOB or XML value is reused on subsequent chunks until the entire LOB or XML is processed.

The LOAD utility also uses LOB streaming for LOBs and XML but only when file reference variables are used.

Whether materialization is reduced, and by how much, depends on the following conditions:

- ▶ JDBC 4.0 and later or ODBC/CLI V8 driver FP4 and later. If using JDBC 3.5, the application has to specify the length of the LOB or XML to be -1.
- ▶ There can be at most one LOB per row for INSERT, UPDATE, or LOAD with file reference variable.
- ▶ There can be at most one XML per row for INSERT or UPDATE with DRDA streaming.
- ▶ There can be one or more number of LOB and XML values per row with INSERT, UPDATE, or LOAD XML with file reference variables and LOB INSERT or UPDATE or using crossloader that require CCSID conversion.
- ▶ For UPDATE, an additional restriction applies in that the UPDATE must qualify just one row where a unique index is defined on the target update column.

This enhancement is available in CM and reduces the virtual storage consumption and the elapsed time. You can also see a reduction in CPU time.

A set of measurements were implemented inserting binary LOBs from a remote client to verify the savings in avoiding materialization (Figure 8-17).

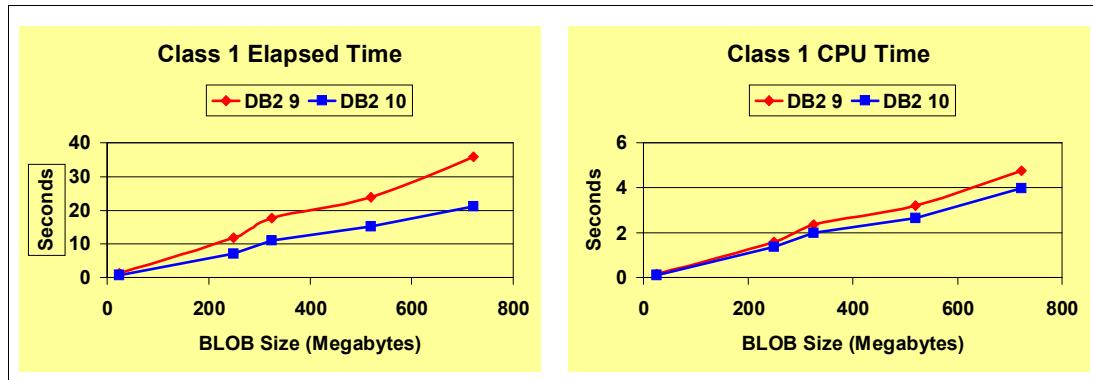


Figure 8-17 Streaming effects with DRDA LOB INSERTS

The improvements are dependent on the size of the LOBs. For a 500 MB LOB we observe a reduction of 30% in elapsed time and 15% in CPU time.

Streaming of LOB and XML data to the DBM1 address space is also available to local applications, for example SQL INSERT UPDATE and the LOAD utility, when file reference variables are used.



Utilities

DB2 10 for z/OS delivers a variety of improvements on utilities to support all new functions, such as universal table space (UTS) enhancements, online schema changes, inline LOBs, XML, and other new functions. One of the more remarkable features is that several utilities integrate the option to use FlashCopy at the table space level, providing possible improvements in availability and performance.

In this chapter, we discuss the following topics:

- ▶ Use of FlashCopy in utilities
- ▶ RUNSTATS
- ▶ RECOVER with BACKOUT YES
- ▶ Online REORG enhancements
- ▶ Increased availability for CHECK utilities
- ▶ REPORT utility output improvement
- ▶ LOAD and UNLOAD
- ▶ DFSORT

9.1 Use of FlashCopy in utilities

With the number of objects that are involved in today's large databases, it has become an overwhelming task to maintain them.

To help in this area, DB2 8 introduced system-level backup and recovery, which uses volume-level FlashCopy. DB2 9 enhanced backup and recovery mechanisms of utilities such as BACKUP SYSTEM, RESTORE SYSTEM, introduced in V8, and RECOVER to assist you in this process. RECOVER was extended in DB2 9 to be able to recover individual objects from system level backups (taken by BACKUP SYSTEM). In addition, the integration of the incremental FlashCopy feature into these utilities minimizes I/O impact and considerably reduces elapsed time for creation of the physical copy. Other DB2 utilities take advantage of the FlashCopy functionality, mostly for availability reasons.

DB2 V8 also introduced CHECK INDEX with option SHRLEVEL CHANGE which increases availability and satisfies the requirement of customers who cannot afford the application outage. It's also dramatically cuts down elapsed time due to parallel processing and usage of FlashCopy V2. Table space are copied on shadow copies where the checking takes place.

DB2 9 added the ability to execute the CHECK DATA and CHECK LOB utilities with the SHRLEVEL CHANGE option resulting in increased availability. This is accomplished by using the IBM FlashCopy V2 feature to maximize performance and accentuate availability.

DB2 10 adds more support and integration of FlashCopy technology at the DB2 object level:

- ▶ The COPY, LOAD, REORG, REBUILD utilities produce an image copy of table space or index using data set level FlashCopy technology and register it in SYSCOPY.
- ▶ When creating a FlashCopy image copy, the following utilities also can create one to four additional sequential format image copies in a single execution:
 - COPY
 - LOAD with the REPLACE option specified
 - REORG TABLESPACE
- ▶ The COPYTOCOPY utility can create sequential format image copies by using an existing FlashCopy image copy as input.
- ▶ The SHRLEVEL CHANGE copy can produce a consistent image copy for multiple tables with no outage, including DB2 catalog and directory. It is also available for other utilities.
- ▶ The RECOVER utility can use a FlashCopy image copy as input dataset.
- ▶ RECOVER can perform prior point in time (PIT) recovery by backing out changes instead of using an image copy and applying log forward.
- ▶ There is more opportunity to use System Level Backup for data set recovery.

DB2 10 limitations when using FlashCopy are as follows:

- ▶ There is no incremental volume level Copy with a FlashCopy imagecopy.
- ▶ The DS8000 does not support incremental data set level FlashCopy.

It is possible to create a system-level backup using BACKUP SYSTEM while in parallel a FlashCopy image copy is taken for an object. However, because the DS8000 does not support cascading FlashCopy (the target volume of a FlashCopy relationship cannot be the source of another FlashCopy relationship), it is not possible to take advantage of data set FlashCopy to recover an object while at the same time the physical background copy of FlashCopy for BACKUP SYSTEM is running (or the volume-level FlashCopy relationship persists due to incremental FlashCopy at volume level).

If the hardware does not support FlashCopy, the system reverts to use IDCAMS Repro to copy the data set.

You can ask DB2 to create a FlashCopy image copy using one of the following options:

- ▶ For the COPY utility:
 - Set DSNZPARM FLASHCOPY_COPY to YES.
 - Specify FLASHCOPY YES or FLASHCOPY CONSISTENT on your COPY utility control statement.
- ▶ For the LOAD utility:
 - FLASHCOPY_LOAD
- ▶ For the REORG utility
 - FLASHCOPY_REORG_TS
 - FLASHCOPY_REORG_INDEX
- ▶ For the REBUILD utility
 - FLASHCOPY_REBUILD_INDEX

The default setting for all these FLASHCOPY DSNZPARMs is NO.

FlashCopy image copy is useful for making image copies of large DB2 objects and reduce CPU utilization; however, using FlashCopy does not mean that you can obtain better performance than sequential image copies with small DB2 objects. For small objects, you get better performance using a system level backup, which avoids the cost of data set allocations.

Creating an image copy by specifying FLASHCOPY CONSISTENT uses more system resources and takes longer than creating an image copy by specifying FLASHCOPY YES, because backing out uncommitted work requires reading the logs and updating the image copy. You cannot specify CONSISTENT when copying objects that have been defined with the NOT LOGGED attribute.

In this section we show the FlashCopy performance for COPY and RECOVER utilities.

9.1.1 COPY utility

In DB2 10, the COPY utility is enhanced to provide an option to use the DFSMSdss fast replication function for taking full image copies by the COPY utility or the inline COPY function of the REORG and LOAD utilities. The DFSMSdss fast replication function invokes FlashCopy to perform the physical data copy operation, which in turn offloads the physical data copy operation to the DS8000 disk subsystem. As a result, no data pages need to be read into the table space buffer pool.

All the measurements used a table space with 1000 partitions, with no indexes. Sliding scale space measurement and DFSMS cannot consolidate the extents. Thus, the first extent was one cylinder, the second extent was two cylinders, and so on. However, DSS generally consolidates the extents when it allocates the FlashCopy data set. This will become more relevant when we discuss RECOVER.

Figure 9-1 shows the accounting report highlights of the utility execution using the FLASHCOPY NO COPY utility option. In the buffer pool activity section, DB2 reads all data pages into the local buffer pool for image copy processing.

HIGHLIGHTS					
PARALLELISM: UTILITY					
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	TOTAL	BPOOL ACTIVITY	TOTAL
ELAPSED TIME	11.381112	0.047218	GETPAGES		100299
CP CPU TIME	0.981997	0.472171	BUFFER UPDATES		58
AGENT	0.022562	0.006620	SEQ. PREFETCH REQS		1564
PAR.TASKS	0.959435	0.465551	PAGES READ ASYNCHR.		100093

Figure 9-1 COPY with FLASHCOPY NO accounting report

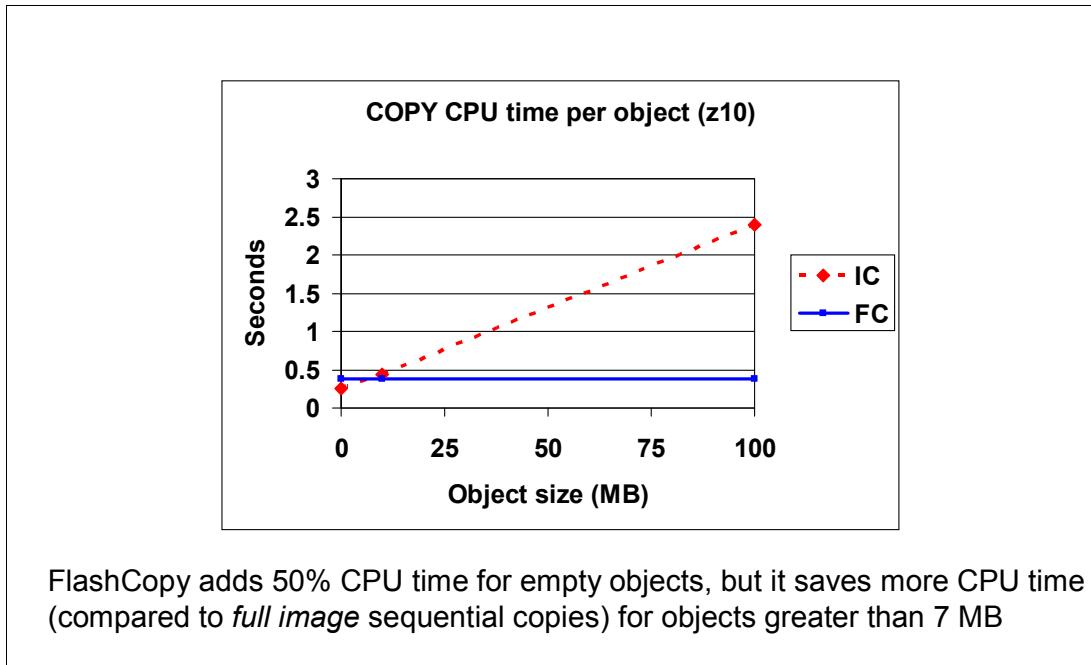
Figure 9-2 shows the accounting report highlights of the utility execution using the FLASHCOPY YES COPY utility option. In the buffer pool activity section, DB2 does not read the data pages that are to be processed into the local buffer pool. Instead, DB2 invokes the DFSMSdss ADRDSSU fast replication function.

HIGHLIGHTS					
PARALLELISM: NO					
TIMES/EVENTS	APPL(CL.1)	DB2 (CL.2)	TOTAL	BPOOL ACTIVITY	TOTAL
ELAPSED TIME	0.731253	0.034548	GETPAGES		103
CP CPU TIME	0.020734	0.004625	BUFFER UPDATES		25
AGENT	0.020734	0.004625	SEQ. PREFETCH REQS		0
PAR.TASKS	0.000000	0.000000	PAGES READ ASYNCHR.		0

Figure 9-2 COPY with FLASHCOPY YES accounting report

The measurements were done on a system z10 processor with 4 engines, DS8300, and z/OS 1.12. Note that z/OS 1.12 has better FlashCopy performance than z/OS 1.11, because DFSMSdss in 1.12 allocates fewer temporary data sets. Measurements were taken for objects with no rows, as well as with both 10 MB objects and 100 MB objects. RMF was used to calculate the overall CPU time, because class 1 CPU time does not capture the DFSMSdss CPU time.

Figure 9-3 shows that on our tests, CPU time increased 50% for small objects, less than 7 MB, whereas FlashCopy is a good option for larger objects.



FlashCopy adds 50% CPU time for empty objects, but it saves more CPU time (compared to *full image* sequential copies) for objects greater than 7 MB

Figure 9-3 Copy CPU time with FlashCopy Yes and No

Figure 9-4 shows that elapsed time is longer for small objects when comparing FlashCopy with sequential image copies. For large objects, FlashCopy is faster than sequential image copies and stays practically constant.

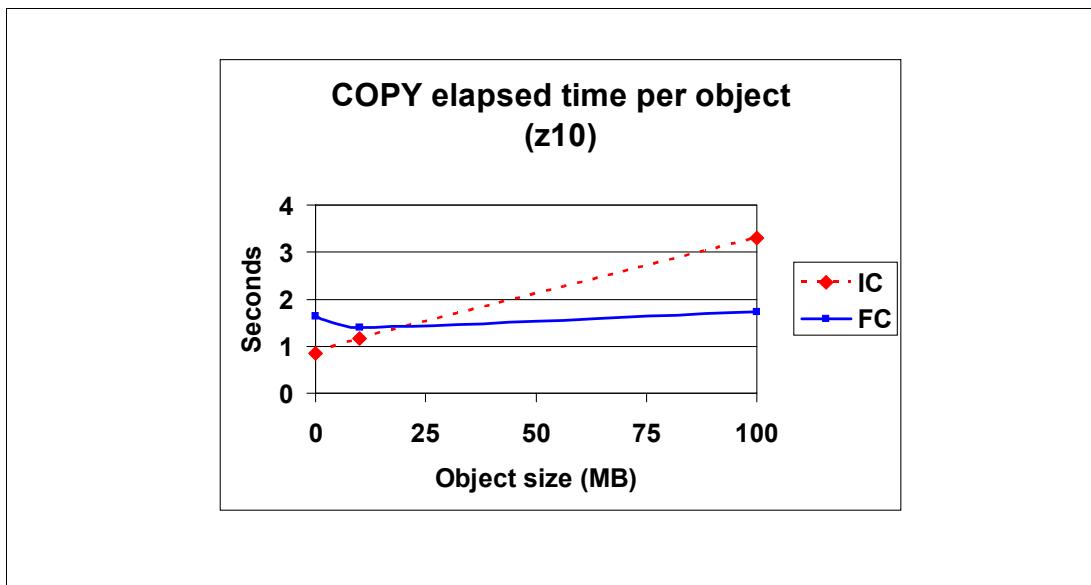


Figure 9-4 Copy elapsed time with FlashCopy Yes and No

From these measurements, we can offer the following considerations:

- ▶ FlashCopy image copy is useful for one of the following cases:
 - Use FlashCopy for objects that need the highest availability for recovery.
 - Use FlashCopy if your image copy cycle window needs reduction.
 - If you use incremental image copies, continue to use them, unless transaction consistent copies are required.
- One use for consistent copies (no outage) is to feed them into DSN1COPY and then move them to a different system.
- ▶ The performance of FlashCopy depends of the size of your table space. Consider the following options:
 - For objects larger than 70 MB (~ 100 cylinders), FlashCopy uses less CPU and elapsed time,
 - For objects between 7 MB and 70 MB, use FlashCopy if CPU constraints are more important than Recover time. Otherwise, use sequential copies.
 - For objects smaller than 7 MB, use sequential image copies or volume FlashCopy
- ▶ If the hardware does not support FlashCopy, the system uses IDCAMS Repro to copy the data set. In this case the only advantage is that Repro does not impact the DB2 buffer pool.

9.1.2 RECOVER utility

During the recovery of an object, if it already exists and both the source and target have the same number of extents, DSS reuses them (no matter what is coded on the RECOVER statement). Otherwise DSS will allocate a new dataset. Figure 9-5 shows that the recovery elapsed time for small objects using sequential image copies is faster than FlashCopy and the recovery using FlashCopy is better for large objects.

The reason why the FlashCopy elapsed time for 10 MB was higher than for a table space with no rows is that DSS needed to scratch and reallocate the table space in order to allocate one big extent that matches the size of the FlashCopy data set.

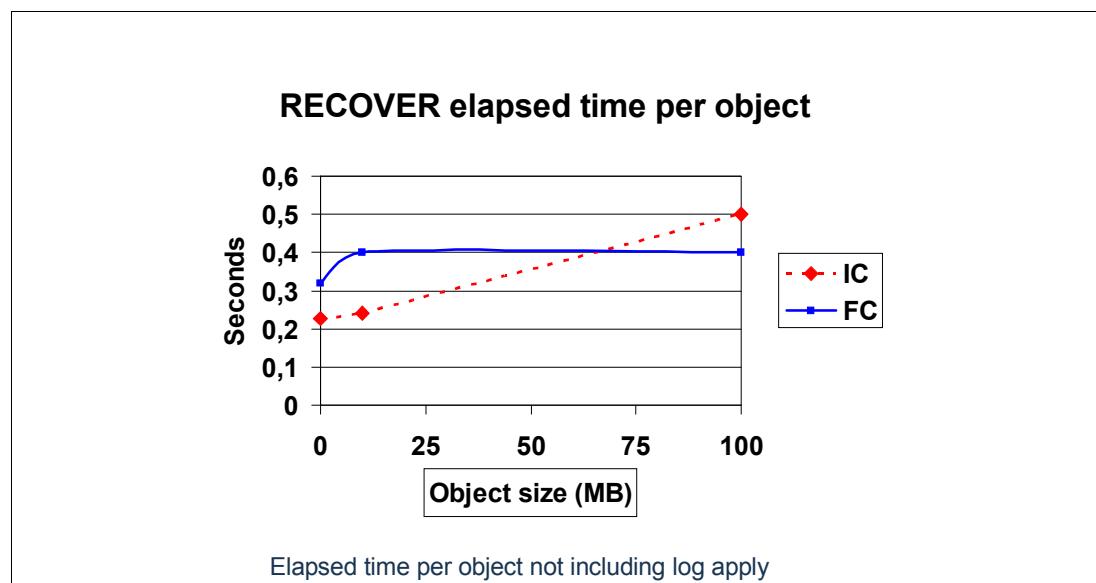


Figure 9-5 RECOVER from FlashCopy

Applying the logs starts before the FCIC background copy is complete. For standard recovery, restore of the image copy must have completed.

9.2 RUNSTATS

With DB2 10, the RUNSTATS utility includes new and changed options:

- ▶ The options USE PROFILE, DELETE PROFILE, SET PROFILE, and UPDATE PROFILE are added to support autonomic statistics. They are described in *DB2 10 for z/OS Technical Overview*, SG24-7892.
- ▶ The KEYCARD option is deprecated. The KEYCARD functionality is now built into the normal execution of the RUNSTATS INDEX utility and cannot be disabled.
- ▶ DB2 10 changes the default for SHRLEVEL from REFERENCE to CHANGE.
- ▶ The TABLESAMPLE SYSTEM option allows RUNSTATS to collect statistics on a sample of the data pages from the table.

In addition, RUNSTATS CPU use can be routed to zIIP engines (see 3.2.1, “RUNSTATS zIIP eligibility” on page 51).

In this section, we look at the new TABLESAMPLE SYSTEM option.

Figure 9-6 shows the syntax for the old and sampling options.

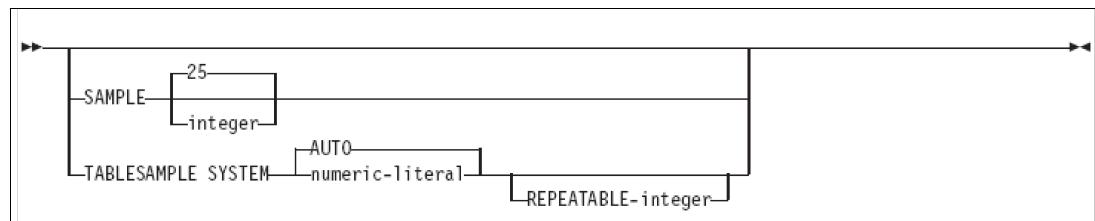


Figure 9-6 RUNSTATS sampling syntax

The SAMPLE option worked at the row level for the specified percentage. The TABLESAMPLE option works at page level, is only valid on *single-table table spaces* (otherwise it reverts to SAMPLE 25) and is not valid for LOB table spaces.

The TABLESAMPLE SYSTEM sampling applied to a table means that each page of the table is included in the sample with probability P/100 and excluded with probability 1 - P/100. For each page that is included, all rows on that page qualify for the sample. Unless the optional REPEATABLE clause is specified, each execution of RUNSTATS will usually yield a different sample of the table.

When AUTO is specified, RUNSTATS will determine a sampling rate based on the size of the table when RUNSTATS is executed. The larger the table, the smaller is the sampling rate. The threshold for sampling is when the table has more than 500,000 rows.

REPEATABLE is specified to ensure that repeated executions of RUNSTATS return the same sample.

Table 9-1 summarizes the possible options.

Table 9-1 TABLESAMPLE options

TABLESAMPLE n	Sampling rate (0.01 to 100)
TABLESAMPLE AUTO	Sampling rate chosen by DB2 (based on table size)
REPEATABLE n	Similar to using the same seed in a random generator to get the same rows sampled again

The RUNSTATS TABLESAMP provides the opportunity to reduce the number of data pages needed to be read in the buffer pool, as we show on Figure 9-7.

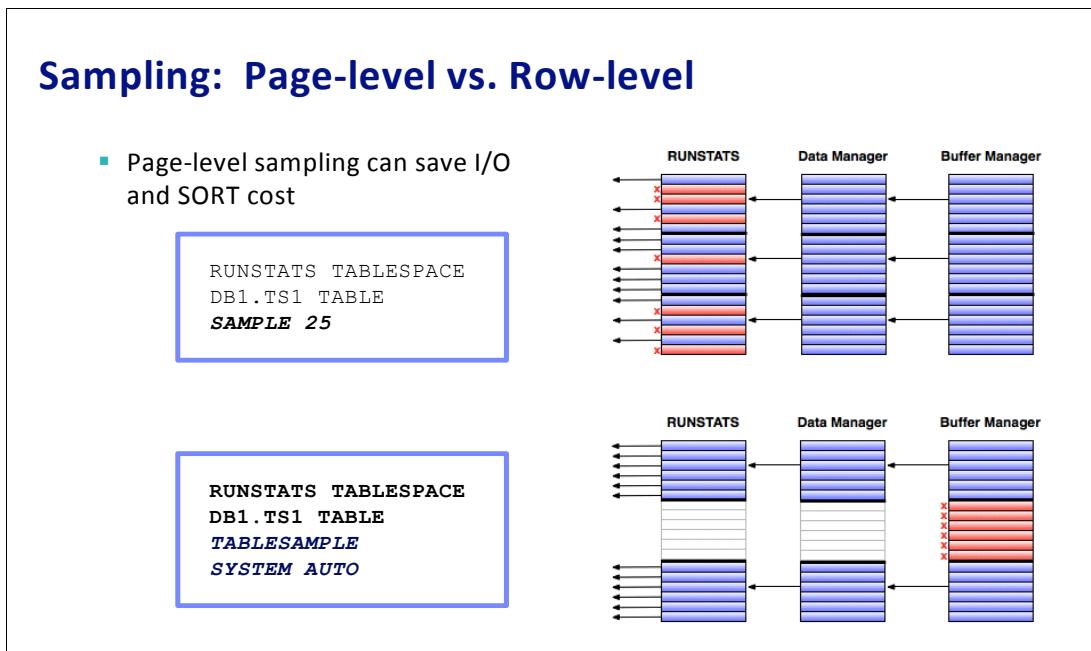


Figure 9-7 Sampling: Page-level versus row-level

Tests were executed on a 30 GB industry standard star schema DW type workload.

In Figure 9-8, we show the results of the execution of a complex RUNSTATS, with COLGROUP and HISTOGRAM, when compared as no sample, SAMPLE 20, and TABLESAMPLE 20. The time is in seconds. The results show a 30% reduction in elapsed time and a 50% reduction in CPU time when comparing row sample level with page sample level.

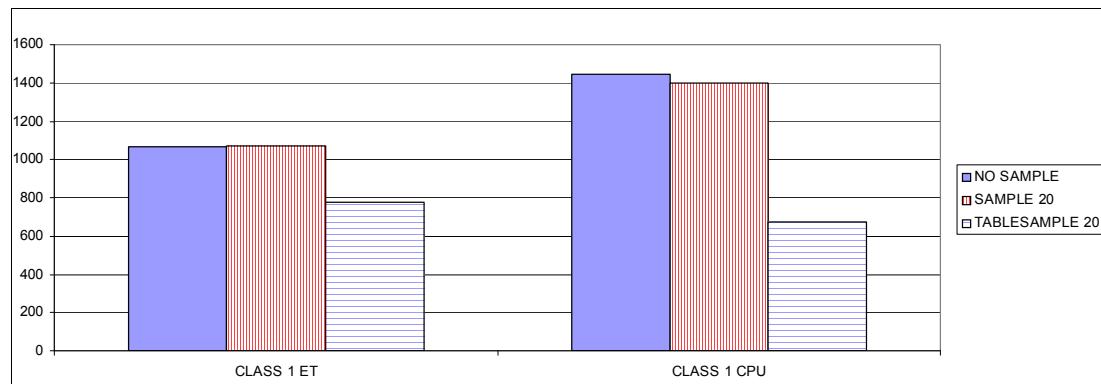


Figure 9-8 Complex RUNSTATS (COLGROUP and HISTOGRAM)

In Figure 9-9, we executed a basic RUNSTATS TABLE ALL INDEX ALL, to compare no sample, SAMPLE 20, and TABLESAMPLE 20. The results (in seconds) show a 25% increase in elapsed time and a 15% reduction in CPU time when comparing row sample level with page sample level.

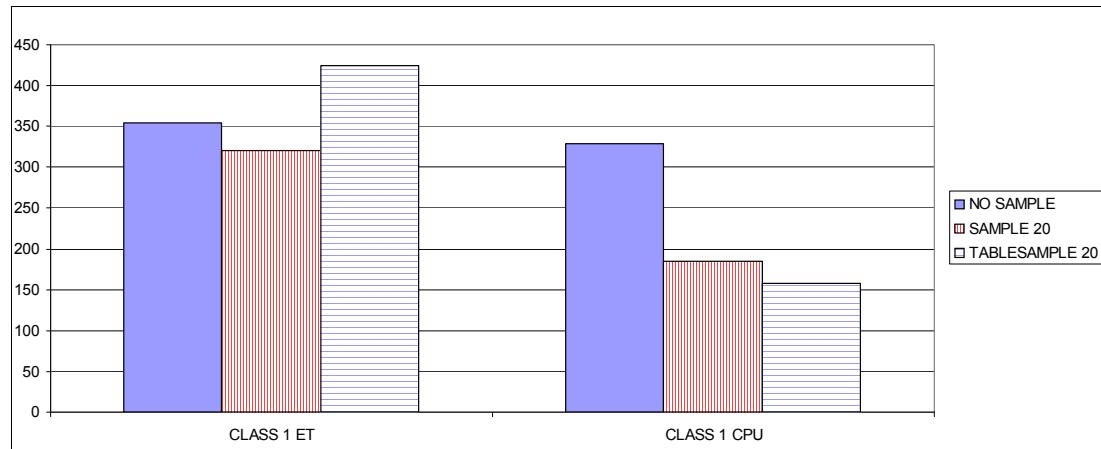


Figure 9-9 Basic RUNSTATS - Sampling 20%

In Figure 9-10, we executed a basic RUNSTATS, TABLE ALL INDEX ALL, to compare no sample, SAMPLE 10, and TABLESAMPLE 10. The results (in seconds) shows a 25% increase in elapsed time and a 20% reduction in CPU time when comparing row sample level with page sample level.

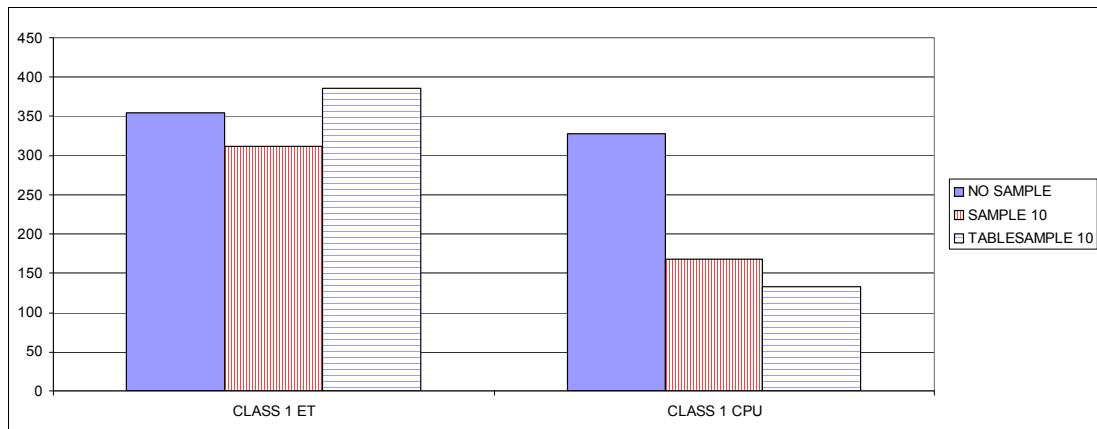


Figure 9-10 Basic RUNSTATS - Sampling 10%

In Figure 9-11, we executed a basic RUNSTATS, table all index all, to compare no sample, sample 5 and TABLESAMPLE 5. The results (in seconds) show a 10% increase in elapsed time and a 20% reduction in CPU time when comparing row sample level with page sample level.

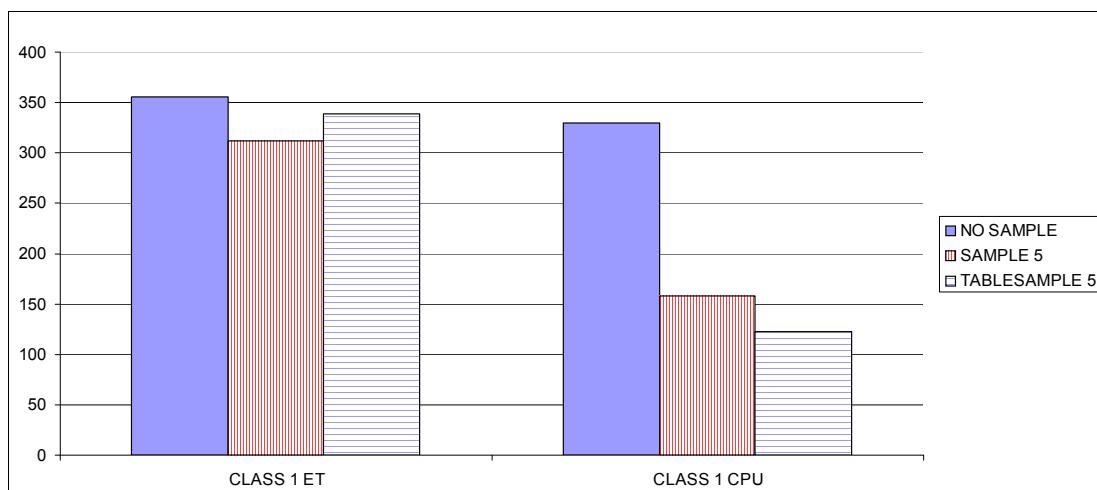


Figure 9-11 Basic RUNSTATS - Sampling 5%

Note that page sampling does sampling for both COLCARD and COLGROUP and row sampling does sampling for COLCARD, but not COLGROUP. Hence, COLGROUP significantly reduces the CPU savings that we get with row sampling.

There is also no sampling for index statistics, with either row or page sampling. Hence index statistics dilute the CPU and elapsed time savings of both row sampling and page sampling. The type of index (that is, partitioning index, non partitioning index (NPI), data partitioned secondary index (DPSI), and so on) can also affect the CPU cost of gathering statistics.

A portion of RUNSTATS can be redirected to a zIIP. The redirection rate varies depending on the RUNSTATS option. It can reach up to 99.9% for RUNSTATS with no additional parameters and it goes down for more complex statistics such as frequency statistics. See 3.2.1, “RUNSTATS zIIP eligibility” on page 51 for details.

9.3 RECOVER with BACKOUT YES

With DB2 9, whether you recover to the current point or a point-in-time recovery, the RECOVER utility identifies the best fitting recovery base, creates a new VSAM cluster, restores the image copy data set, then always reads forward the SYSIBM.SYSLGRNX to find the RBA ranges during which the cluster was involved in updates, and applies the log records. This activity can take a significant amount of elapsed time and resources depending on amount of changes to *apply*.

With DB2 10, RECOVER can alternatively *back out* the changes starting from the currently operational cluster. You specify the RECOVER TABLESPACE statement with the BACKOUT YES option; in this case RECOVER does not require an image copy.

DB2 performs the following steps to handle the BACKOUT YES request:

- ▶ A prior point in time is specified to be the recovery point.
- ▶ DB2 identifies the latest checkpoint that occurred prior to the specified point in time.
- ▶ DB2 starts the LOGCSR phase, which rebuilds the current status in order to identify open units of recovery that need to be handled as part of the process.
- ▶ With this knowledge, DB2 can now back out data up to the RBA that you specified as the point in time.
- ▶ At the end, uncommitted units of recovery at the BACKOUT log point are backed out to make the object consistent.

Attention: After running the RECOVER BACKOUT YES, you cannot run any subsequent recover with backout if the RBAs specified are smaller than the START_RBA of the latest BACKOUT YES recovery that you ran. An attempt to do so ends with an error message.

However, if you have to restore data to an earlier point in time, you can execute the regular DB2 recovery process.

9.4 Online REORG enhancements

When we discuss online REORG, we mean REORG....SHRLEVEL REFERENCE or REORG.... SHRLEVEL CHANGE.

We examine two enhancements:

- ▶ REORG for base tables spaces with LOBs
- ▶ Online REORG and prefetch

9.4.1 REORG for base tables spaces with LOBs

This section shows the performance of the new online REORG for base table spaces with LOBs functions introduced on DB2 10.

In DB2 9, when REORG is run against a partition-by-growth table space with LOBs, the data records from part *n* before REORG need to go back into part *n* after REORG. Thus, there is no data movement across partitions when LOB is present, which can lead to REORG failure due to space issues when PCTFREE or FREEPAGE are set. This has been partially addressed in DB2 9 by APAR PK83397 which introduced DSNZPARM REORG_IGNORE_FREESPACE.

DB2 10 resolves this issue with the AUX YES support and therefore no longer supports the DSNZPARM REORG_IGNORE_FREESPACE. The AUX keyword allows you to specify whether DB2 reorganizes the associated auxiliary LOB table spaces along with the base table space. Data movement across partitions during REORG on partition-by-growth with LOB is possible by means of REORG moving the corresponding LOB data. REORG DISCARD, REORG BALANCE, and ALTER LIMITKEY are also possible.

When AUX YES is specified, DB2 fetches the row ID for each row as the rows are unloaded and it then probes the auxiliary index(es) to unload the corresponding LOB or LOBs. In contrast, if you REORG the LOB table space by itself, DB2 does an index scan of the auxiliary index and unloads the LOBs in row ID sequence.

Thus, although the result of using AUX YES is almost the same as though you separately reorganized the base table space, there is a performance cost of being able to move rows from one partition to another. This performance cost can be mitigated by using a large buffer pool for the auxiliary indexes separate from the LOB table spaces, as shown in Figure 9-12. If the performance cost is unacceptable to you and if you do not need to move rows across partitions, then you might want to explicitly specify AUX NO.

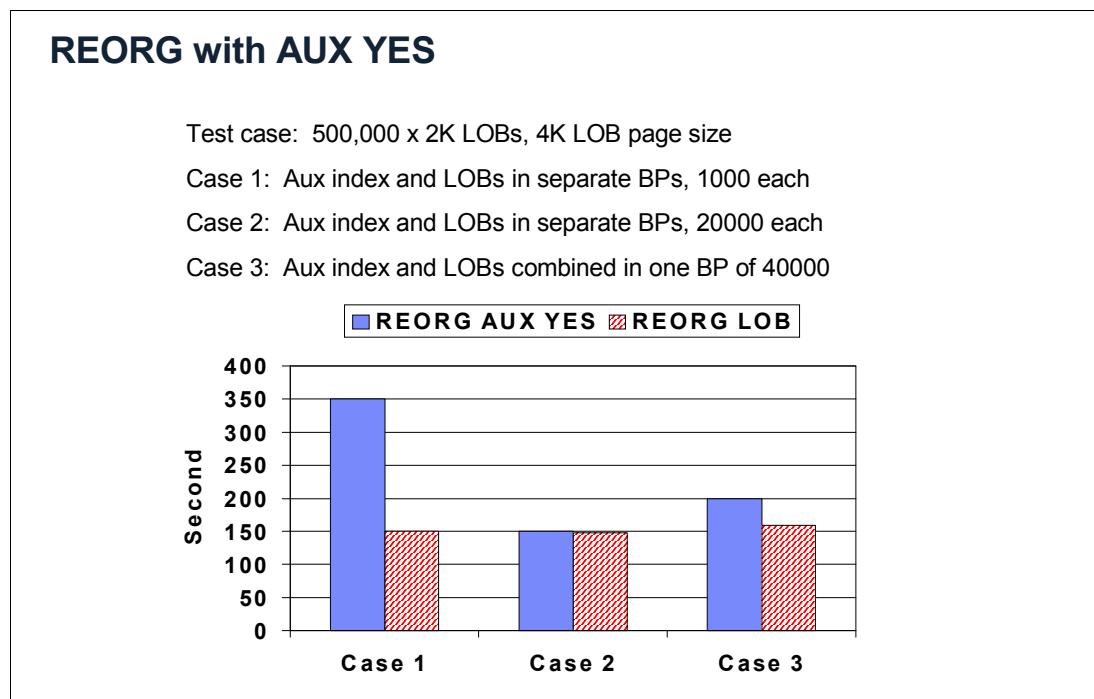


Figure 9-12 REORG LOB AUX YES

Tip: When you create persistently large LOB objects, it is common to specify the NOT LOGGED parameter for the LOB table space to avoid impacting the log. However, in order for DB2 to reorganize LOBs with SHRLEVEL CHANGE, it is necessary to turn on full logging for all LOB objects for the duration of the REORG utility run. This is done automatically by DB2. You do not have to change anything for the table space definition, but you will note an increase in log volume when executing REORG SHRLEVEL CHANGE AUX YES.

REORG with AUX YES might perform worse than a concurrent REORG of the base table space with AUX NO and the LOB table space.

There is a performance cost of being able to move rows from one partition to another. If you do not need to move rows across partitions; you might want to explicitly specify AUX NO.

9.4.2 Online REORG and prefetch

This section shows the performance of the online REORG partition when we have disorganized NPIs comparing DB2 10 to DB2 9.

In DB2 9, when online REORG is run against a partition of table space performance degrades if the NPI is disorganized because of a high number of synchronous read I/O when the utility unloads the whole NPI to its shadow data sets.

In DB2 10, REORG solves this problem using the same solution described in 2.3.1, “Disorganized index scan using list prefetch” on page 27 for queries. If the NPI is organized, the utility uses dynamic prefetch. If the index is disorganized, DB2 uses list prefetch. This change results in a significant elapsed time reduction with some CPU improvement. Both REORG of a partition and the REORG INDEX are improved.

The following measurements were executed on DB2 9 and DB2 10 in a non-data sharing running on z/OS R1.12 and DS8300 disks. We ran REORG TABLESPACE PART SHRLEVEL(CHANGE) and REORG INDEX (NPI) SHRLEVEL(CHANGE) on a classic partition table space of 100 M rows, 20 partitions and 11.8 GB.

Table 9-2 shows 7% CPU and 15% elapsed time improvement in DB2 10 for partition level REORG compared to DB2 9.

Table 9-2 Performance of REORG TABLESPACE PART SHRLEVEL(CHANGE)

REORG Part 1	DB2 9		DB2 10	
	CPU	ET	CPU	ET
Unload	0.08	6.87	0.08	6.57
Reload	8.25	483	8.78	331
Unload NP1	58.14	483	24.16	331
Sort	8.57	509	9.08	512
Build	247	509	250	512
Total	342	1004	318	852

Table 9-3 shows 36% class 3 times improved on DB2 10 and the number of synchronous read I/O in unload is significantly reduced.

Table 9-3 Performance of REORG TABLESPACE PART SHRLEVEL(CHANGE) detail

REORG Part 1	DB2 9		DB2 10	
	Class 3 time	No. events	Class 3 time	No. events
Class 3 sync I/O	998	4,338,399	304	1,277,821
Class 3 other read sync I/O	28	27,872	362	142,733

REORG Part 1	DB2 9		DB2 10	
	Class 3 time	No. events	Class 3 time	No. events
Time	1,114		716	
	NPI 1	NPI 2	NPI1	NPI 2
Get pages	325,070,000	6,014,954	325,019,000	6,022,498
Updates	20,502,204	8,574,747	20,502,204	8,574,747
Sync read	3,159,089	1,179,256	1,270,111	7,673
Seq. prefetch	0	0	0	0
List prefetch			246,034	108,601
Dyn. prefetch	3,905	27,352	6,160	27,353
Async. Read	118,739	875,111	2,015,227	2,054,260

Table 9-4 shows DB2 10 REORG INDEX improved 27% in CPU time and 72% in elapsed time compared to DB2 9.

Table 9-4 Performance of NPI - REORG INDEX SHRLEVEL(CHANGE)

REORG NPI 1	DB2 9		DB2 10	
	CPU	ET	CPU	ET
Unload	39.95	475	19.32	54
Build	28.11	99	30.47	104
Total	68.08	576	49.81	159
	Class 3 time	No. events	Class 3 time	No. events
Class 3 database sync I/O	441	1,968,650	2.60	8,340
Class 3 other read sync I/O	0	0	20.37	50,321
REORG NPI 2	CPU	ET	CPU	ET
Unload	25.75	284	14.14	37
Build	23.02	67	25.06	60
Total	49	354	39.22	97
	Class 3 time	No. events	Class 3 time	No. events
Class 3 database sync I/O	264	1,179,172	2.18	7,460
Class 3 other read sync I/O	0	0	18.60	35,517

Table 9-5 shows that in DB2 9 online REORG of a disorganized NPI performance, there was a high number of sync read I/O. Instead, DB2 10 uses list prefetch during the unload of the NPI and reduces the number of sync read I/O.

Table 9-5 Performance of NPI - REORG INDEX SHRLEVEL(CHANGE) details

REORG INDEX	DB2 9		DB2 10	
	NPI 1	NPI 2	NPI 1	NPI 2
Get pages	6,181,161	3,998,116	6,190,119	4,005,825
Updates	5,586,746	3,740,000	5,586,746	3,740,000
Sync. read	1,968,639	1,179,168	8,334	7,458
Seq. prefetch	0	0	0	0
List prefetch	0	0	246,011	108,590
Dyn. prefetch	0	0	0	0
Async. read	0	0	1,968,699	1,179,060

In conclusion, DB2 10 online REORG improves elapsed time significantly on very disorganized NPIs. The performance can also vary depends on:

- ▶ The degree of disorganization of the NPI
- ▶ The index key size
- ▶ The overall size of the NPI
- ▶ The number of NPIs

There is less frequent need to REORG the NPI before reorganizing the table space partition.

9.5 Increased availability for CHECK utilities

Prior to DB2 10, when you run a CHECK DATA or CHECK LOB utility and DB2 detects a referential integrity violation, it places the dependent object into check pending (CHKP) restrictive state. As a result, the table space which was fully accessible before, is completely restricted for any SQL access.

With DB2 10 CM mode, the utilities CHECK DATA and CHECK LOB have been changed. When you run these utilities with the new functionality turned on, DB2 no longer sets the CHKP status for table space or LOB objects when you run those utilities and DB2 finds inconsistencies. DB2 only reports the problem and helps you obtain details about possible inconsistencies. This function is turned on automatically but can be turned off by setting the new DSNZPARM CHECK_SETHOOK to YES (NO by default).

9.6 REPORT utility output improvement

In DB2 9, if you turn on the functionality to use system level backups for object level recoveries, DSNZPARM SYSTEM_LEVEL_BACKUPS = YES, DB2 searches the information available in SYSIBM.SYSCOPY and BSDS during RECOVER and picks the most recent recovery base as basis for your recovery. If you want to know before you start the RECOVER utility whether a recent image copy or a SLB will be used for an upcoming recovery, you must manually compare the information in SYSIBM.SYSCOPY or the output of the REPORT RECOVERY output with the BACKUP SYSTEM summary section in the DSNJU004 (print log map) utility output. The REPORT RECOVERY utility does not consider SLBs in DB2 9 for z/OS.

With DB2 10, the REPORT RECOVERY utility checks for the setting of DSNZPARM SYSTEM_LEVEL_BACKUPS. If SYSTEM_LEVEL_BACKUPS is set to YES, the REPORT RECOVERY utility considers all sorts of copy resources and lists them in its job output, allowing a better choice in your recovery execution.

In addition to the DSNZPARM setting and independent from it, the REPORT RECOVERY utility has a new section at the bottom of the job output, which shows information about system level backups that exist or used to exist for this specific DB2 subsystem.

For more information about the REPORT RECOVERY utility output, see *DB2 10 for z/OS Technical Overview*, SG24-7892.

9.7 Utility BSAM enhancements for extended format data sets

z/OS 1.9 introduced support for long-term page fixing of basic sequential access method (BSAM) buffers, and z/OS 1.10 introduced support for 64-bit BSAM buffers if the data set is extended format. DB2 10 utilities exploit these recent z/OS enhancements by offering the following enhancements:

- ▶ Allocating 64-bit buffers for BSAM data sets
- ▶ Allocating more BSAM buffers for faster I/O
- ▶ Long term page fixing BSAM buffers
- ▶ DB2 10 utilities, increase MULTSDN from 6 to 10 and MULTACC from 3 to 5

All the DB2 10 utilities using BSAM benefit from this change. For example, Copy and Unload are faster in elapsed time. Copy, Unload, Load, and Recover all have less CPU time than they have without these enhancements.

Tip: If you currently explicitly specify BUFNO on your data set allocations, your value overrides MULTSDN, so nothing changes.

9.8 LOAD and UNLOAD

In this section, we discuss the following topics:

- ▶ LOAD and UNLOAD with spanned records
- ▶ LOAD and UNLOAD internal format
- ▶ LOAD PRESORTED

9.8.1 LOAD and UNLOAD with spanned records

Prior to DB2 8, DB2 sometimes was not able to LOAD or UNLOAD large LOB or XML columns with other non-LOB or XML columns into the same data set because the I/O record size was limited to 32 KB for VB type data sets.

DB2 V8 and DB2 9 allow loading or unloading of LOB or XML larger than 32 KB of data from or into separate data sets using file reference variables (FRV), but the UNLOAD utility's support is restricted to partitioned data sets and UNIX file systems. Although SQL can use FRV for sequential files, LOAD and UNLOAD cannot. That also means that the utilities cannot use tapes with FRV because all tape data sets are sequential.

Furthermore, partitioned data sets and UNIX file systems are slower than a traditional sequential SYSREC file, especially for UNLOAD, because of the overhead to update the directory of the data set or file system, and the block size (except for a PDS) is not optimal for managing large objects. Partitioned data sets also have the limitation that they support a maximum number of members. DB2 cannot unload more rows from the table space with LOB data than this number.

DB2 10 introduces support for spanned records (RECFM = VS or VBS) in LOAD or UNLOAD; and now LOB columns and XML columns of any size can be loaded or unloaded from the same sequential SYSREC data set with other non-LOB columns. Spanned records overcome the limitations of FRV and all of the LOB or XML data of a given table space or partition can be written to a single sequential file, which can reside on tape or disk and can span multiple volumes.

Figure 9-13 shows the performance of unloading LOB using VBS format, which has much better performance when compared with PDSE format.

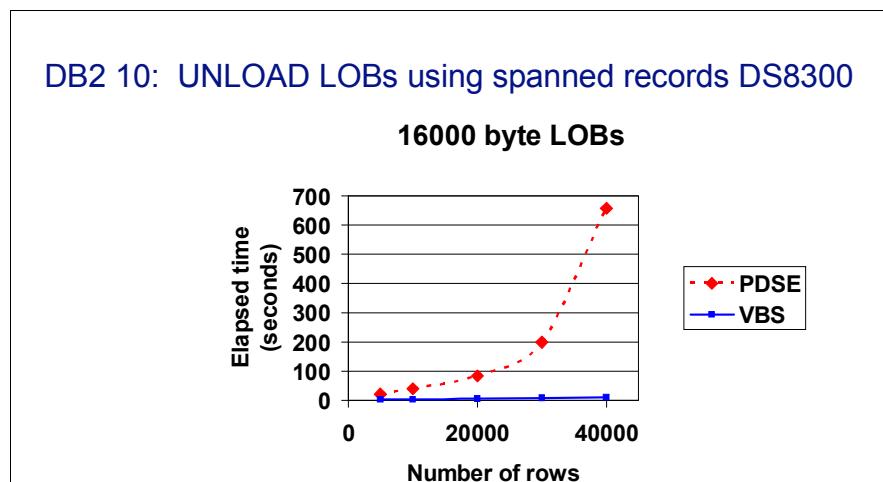


Figure 9-13 UNLOAD LOB using spanned records

100,000 rows took 2 hours, 55 minutes using PDSE and 21 minutes with PDS. VBS is orders of magnitude faster than PDSE or PDS. However, Load/Unload elapsed time is a function of the percentage of non-null LOBs.

Figure 9-14 shows the performance of UNLOAD of LOB using VBS format, compared with zFS, and HFS format. As we can see, VBS has a good performance: VBS is 80% faster than HFS for small LOBs.

LOAD and UNLOAD XML or LOB columns data on VBS format seem to be a good option to overcome the limitations of FRV. All the LOB or XML data of a given table space or partition can be written to a single sequential file.

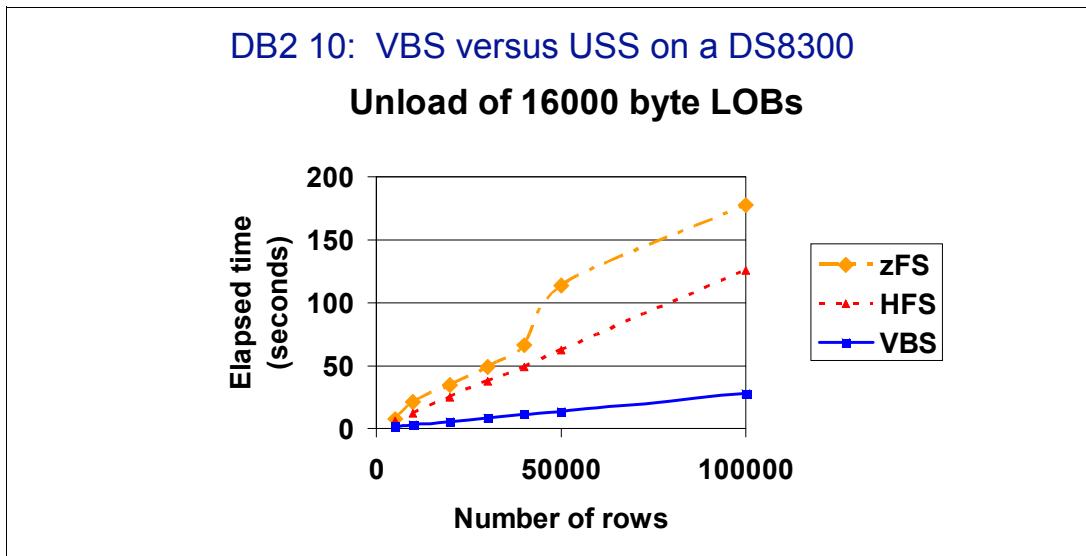


Figure 9-14 VBS versus USS

9.8.2 LOAD and UNLOAD internal format

The LOAD and UNLOAD internal format was introduced by PM19584 into DB2 9 and DB2 10. The whole row is unloaded and loaded in internal format, which avoids column level CPU processing. This saving can be diluted by COMPRESS tablespace or indexes build. These features are not supported for LOBs, XML, generated columns.

Figure 9-15 shows that the performance of UNLOAD and LOAD in an internal format is much better than external format. We executed UNLOAD and LOAD using all defaults with no column specifications, no indexes, no WHEN clause, non-padded input, non-padded output, and no statistics.

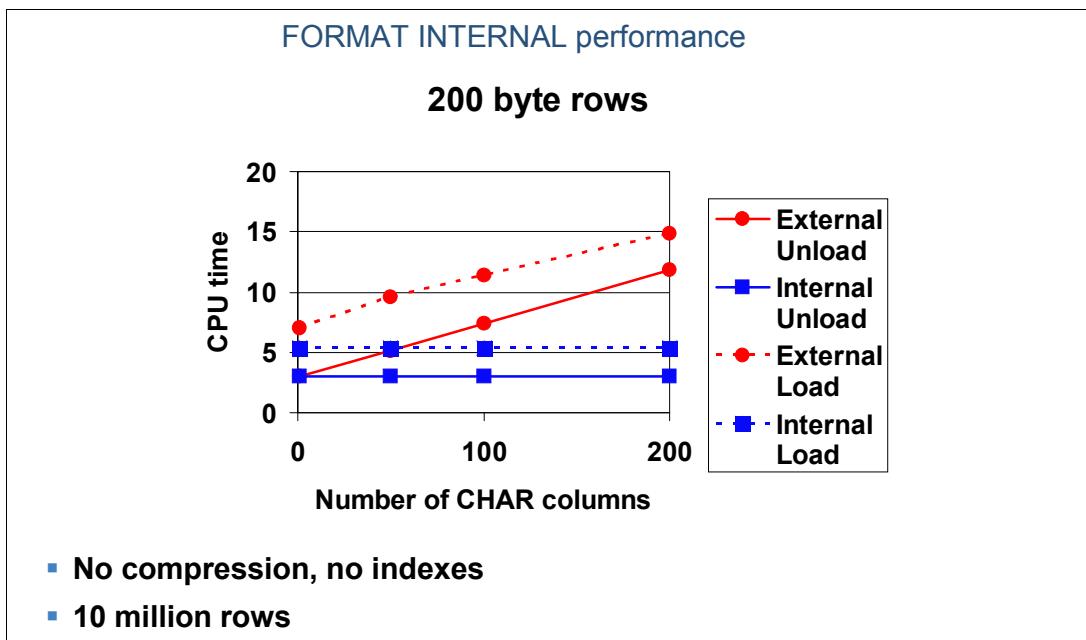


Figure 9-15 LOAD and UNLOAD internal format

Figure 9-16 illustrates the effect of using internal format for a real customer table that has hundreds of columns consisting mostly of VARCHAR, with some DECIMAL columns. For this case, performance was measured with and without compression. We observe that compression tends to dilute the CPU savings, especially for LOAD.

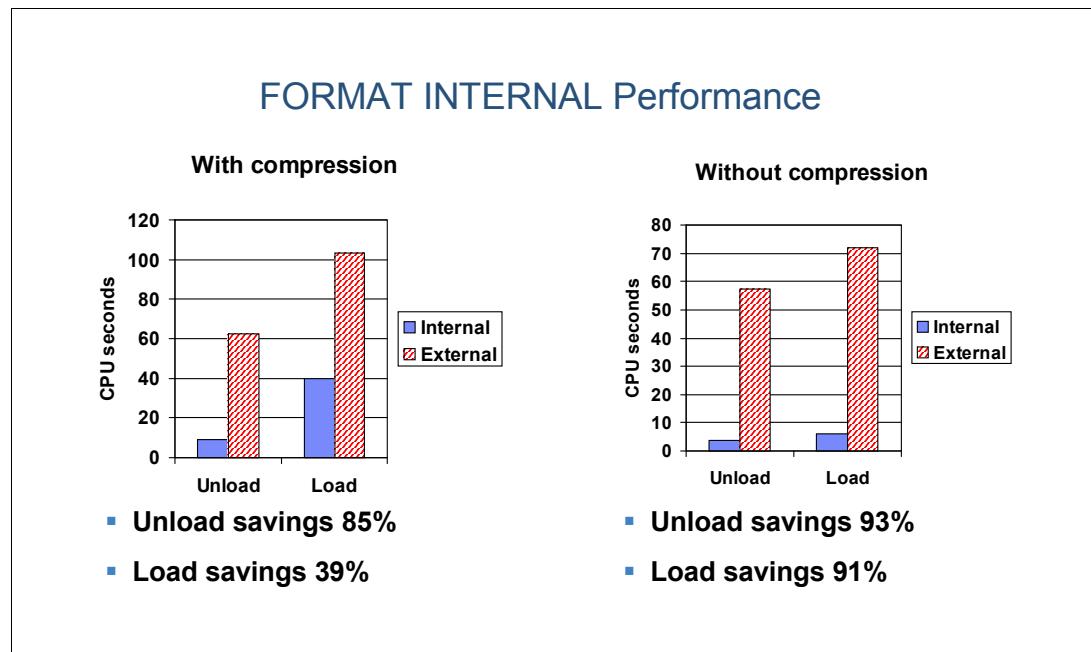


Figure 9-16 LOAD and UNLOAD internal format with compress

In conclusion, LOAD and UNLOAD internal format show good performance, but it can be diluted by adding compression, statistics, and indexes build.

9.8.3 LOAD PRESORTED

LOAD PRESORTED was introduced by PM19584 (same APAR as INTERNAL FORMAT) into DB2 9 and DB2 10 and it improves the performance of loading data when the input data is already in sorted clustering key order. When PRESORTED YES is specified, LOAD skips the SORT of the clustering key index to improve performance. This is a good option if you already presorted the input data in clustering key order prior to running the LOAD utility.

In Table 9-6 we demonstrate the performance of the LOAD PRESORTED YES option running on a UTS table of 50 million rows, 10 partitions. It shows significant improvement in CPU and elapsed time, which was observed mostly during the SORT phase.

Table 9-6 LOAD PRESORTED YES option

	DB2 9		Presorted		% DELTA	
	CPU	Elapsed time	CPU	Elapsed time	CPU	Elapsed time
no index TOTAL	48.23	95.39	48.37	96.09	0%	1%
1 index TOTAL	69.77	126.49	69.28	102.96	-1%	-19%
SORT - 1 index			0.01	0.167		
2 indexes TOTAL	123	123	110	121	-11%	-2%

	DB2 9		Presorted		% DELTA	
SORT - 2 indexes	46.68	42.58	33.45	37.33	-28%	-12%
3 indexes TOTAL	163.1	130.2	148.4	130.9	-9%	1%
SORT - 3 indexes	73.26	45.72	57.15	38.69	-22%	-15%
6 indexes TOTAL	301	166	285	164	-5%	-2%
SORT - 6 indexes	180.24	53.15	163.59	43.63	-9%	-18%

If you already presorted the input data in clustering key before running the LOAD utility, use LOAD PRESORTED YES to skip the SORT of the clustering key index.

9.9 DFSORT

DB2 utilities invoke DFSORT to sort in the following cases:

- ▶ Index and foreign keys (LOAD, REORG, REBUILD, CHECK INDEX, CHECK DATA)
- ▶ Data rows (REORG with SORTDATA)
- ▶ LOB info (CHECK LOB)
- ▶ RUNSTATS DPSI and aggregation statistics and RUNSTATS distribution statistics.

The sort phase of these utilities is often the most critical phase and becomes the main bottleneck for their execution time, especially when tables have many indexes, accounting for up to 75% of the total CPU.

Recent releases of z/OS have improved the functions of DFSORT.

We briefly describe a zIIP enhancement and a performance enhancement.

9.9.1 DFSORT additional zIIP redirect

In z/OS 1.11, DFSORT is also modified to allow additional zIIP redirect for DB2 utilities in case of in-memory object sort operations of fixed length records. This enhancement is included in the z/OS 1.11 base and is delivered to z/OS 1.10 through APAR PK85856. This feature is included in the DB2 10 base and requires DB2 APAR PK85889 to be installed to function in DB2 for z/OS version 8 and 9. When the additional zIIP redirect takes place, DFSORT issues message ICE256I:

ICE256I DFSORT CODE IS ELIGIBLE TO USE ZIIP FOR THIS DB2 UTILITY RUN

This enhancement applies to DB2 utility operations that invoke DFSORT for fixed-length record sort processing (for example, index key sort). Be aware that this enhancement does not apply to REORG data sorts because these sorts involve variable-length record sort processing. The sort record type is indicated by the DFSORT runtime message ICE201I:

ICE201I G RECORD TYPE IS V - DATA STARTS IN POSITION 5

In this case, message ICE201I indicates that a variable length record type sort was performed, which does not provide any additional DFSORT zIIP eligibility.

9.9.2 DFSORT performance enhancements

DFSORt APAR PM18196 introduces some performance enhancements. To demonstrate the effect of these DFSORT enhancements on the DB2 utility, we executed LOAD, REORG, REBUILD, CHECK INDEX, and RUNSTATS with COLGROUP with SORTNUM elimination¹, on a table of 100 M rows, 20 partitions, and 6 indexes.

Figure 9-17, when we include PM18196 on z/OS 1.10, shows up to 41% total CPU reduction, up to 57% CPU reduction in SORT, and up to 18% elapsed time reduction.

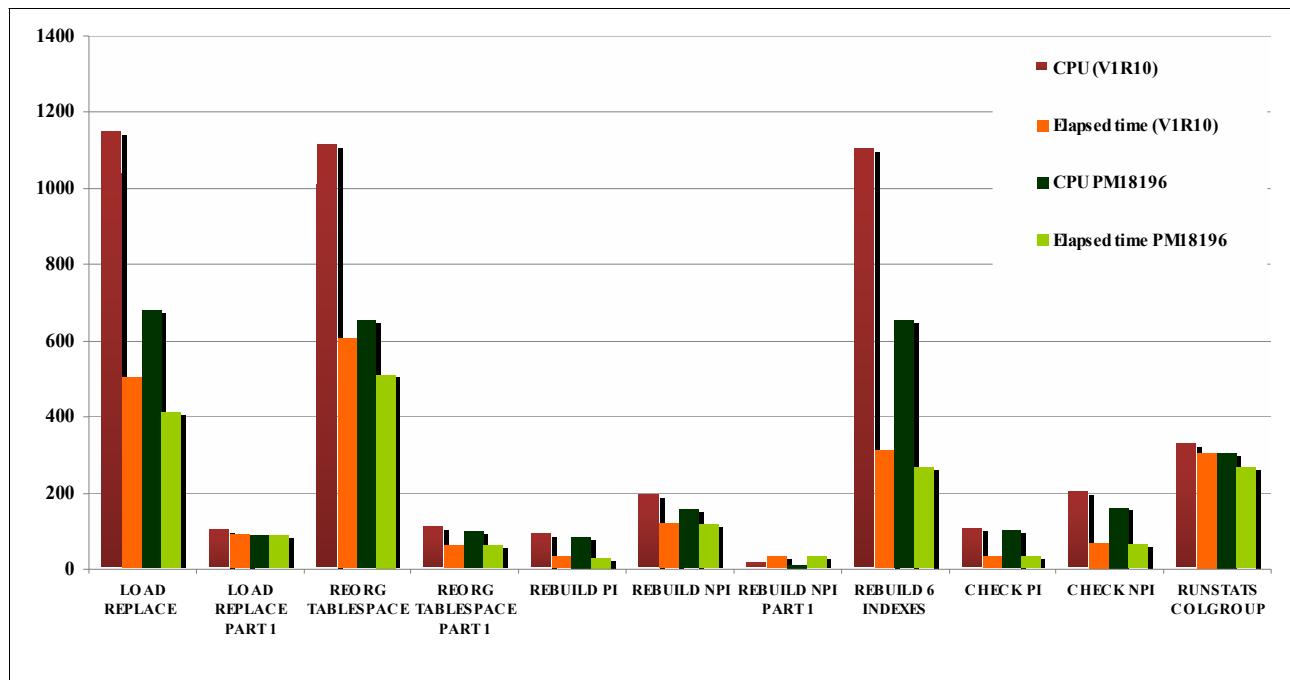


Figure 9-17 DFSORT V1R10 PM18196 and no zIIP

¹ "SORTNUM elimination" refers to the use of DSNZPARM UTSORTAL=YES.

Figure 9-18 shows that, when we include PM18196 and 1 zIIP engine, we have up to 37% total CPU reduction and up to 12% elapsed time reduction, and only a small amount of zIIP reroute.

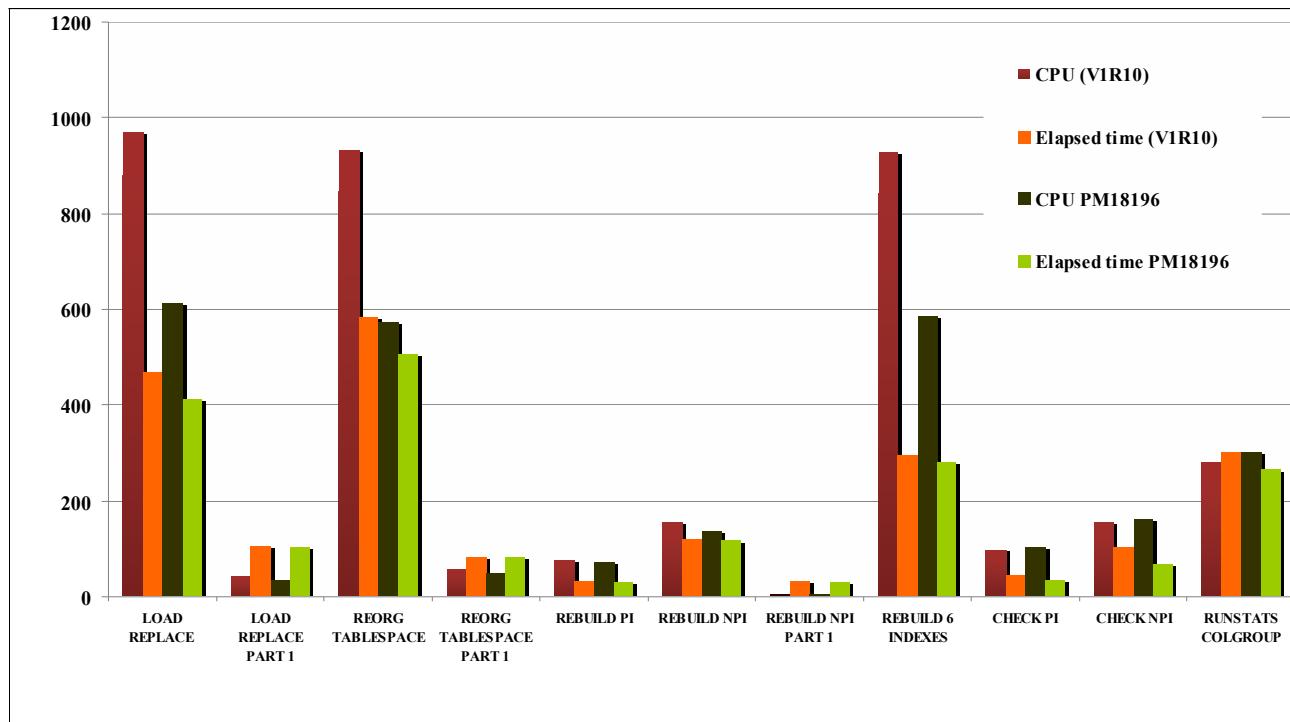


Figure 9-18 DFSORT V1R10 PM18196 and 1 zIIP

LOAD, REORG, REBUILD, CHECK INDEX, CHECK DATA, CHECK LOB, and RUNSTATS invoke DFSORT to sort. The new releases of z/OS have improved the CPU performance of DFSORT. Here is a summary of the changes:

- ▶ V1R10 DFSORT prefers Memory Object sort, which is high CPU intensive, but it is zIIP enabled.
- ▶ V1R10 DFSORT with PM18196 favors Hipersorting, which is not zIIP enabled, but much less CPU expensive.
- ▶ V1R12 DFSORT enhances sort CPU further by favoring the less CPU intensive path and enables it to use memory objects instead of Hiperspaces as work space.



Security

For regulatory compliance reasons or accountability, auditability, increased privacy, and security requirements, many organizations focus on security functions when designing their IT systems.

DB2 10 for z/OS provides a set of options that improve and further secure access to data held in DB2 for z/OS to address these challenges:

- ▶ Policy-based audit capability
- ▶ More granular system authorities and privileges
- ▶ System-defined routines
- ▶ The REVOKE dependent privilege clause
- ▶ Support for row and column access control
- ▶ Support for z/OS security features

In this chapter we discuss the performance implications related to these topics:

- ▶ Policy-based audit capability
- ▶ Support for row and column access control
- ▶ Recent maintenance notes

10.1 Policy-based audit capability

DB2 provides a variety of authentication and access control mechanisms to establish rules and controls. However, to protect against and to discover and eliminate unknown or unacceptable behaviors, you need to monitor data access. DB2 10 assists you in this task by providing a powerful and flexible audit capability that is based on audit policies and categories, helping you to monitor application and individual user access, including administrative authorities. When used together with the audit filtering options introduced in DB2 9 for z/OS, policy-based auditing can provide granular audit reporting. For example, you can activate an audit policy to audit an authorization ID, a role, and DB2 client information.

The auditing capability is available in DB2 10 new-function mode.

10.1.1 Audit policies

An *audit policy* provides a set of criteria that determines the categories that are to be audited. Each category determines the events that are to be audited. You can define multiple audit policies based on your audit needs.

Each policy has a unique name assigned, which you use when you complete the following tasks:

- ▶ Create an audit policy by inserting a row into the SYSIBM.SYSAUDITPOLICIES table
- ▶ Enable an audit policy by issuing a START trace command
- ▶ Display the status of an activated audit policy by issuing a DISPLAY TRACE command
- ▶ Disable an audit policy by issuing a STOP TRACE command

There are several security events that must be audited to comply with laws and regulations and to monitor and enforce the audit policy. For example, auditing is usually performed for the following events:

- ▶ Changes in authorization IDs
- ▶ Unauthorized access attempts
- ▶ Altering, dropping, and creating database objects
- ▶ Granting and revoking authorities and privileges
- ▶ Reading and changing user data
- ▶ Running DB2 utilities and commands
- ▶ Activities performed under system administrative and database administrative authorities

DB2 10 groups these events into the following *audit categories*. The same categories were first introduced in DB2 for Linux, UNIX, and Windows.

- ▶ CHECKING
- ▶ CONTEXT
- ▶ DBADMIN
- ▶ EXECUTE
- ▶ OBJMAINT
- ▶ SECMAINT
- ▶ SYSADMIN
- ▶ VALIDATE

You create an audit policy by inserting a row into the SYSIBM.SYSAUDITPOLICIES catalog table. Each row in SYSIBM.SYSAUDITPOLICIES represents an audit policy. An audit policy is uniquely identified by its policy name.

APAR PM28296 has introduced some enhancements to audit policies capabilities:

- ▶ While audit policies provide the ability to monitor application and individual user access, including administrative authorities, security administrators with SECADM authority can create those security policies and collect audit traces. Before this change, any user with TRACE privilege has been able to stop the audit traces using STOP TRACE command. This can be a security vulnerability.

The APAR introduced a new value 'S' to the DB2START column of the SYSIBM.SYSAUDITPOLICIES table, which enables audit trace during the DB2 startup and can only be stopped by a user with SECADM authority or DB2 shutdown. If the audit policies have already been started, then after updating the DB2START column, the trace needs to be stopped and restarted to get the secure behavior.

- ▶ The SECMAINT category had been changed to include IFCID 271, which audits row permission and column mask DDL.

For details of the audit policies tasks and audit categories, see *DB2 10 for z/OS Technical Overview*, SG24-7892, and *DB2 10 for z/OS Administration Guide*, SC19-2968.

Audit trace and audit categories

Table 10-1 shows the mapping between audit trace classes, the instrumentation facility component identifiers (IFCIDs), and the new audit categories. The previous audit classes and the new audit policies show different auditing functionality and are grouped differently.

Table 10-1 Mapping of audit class to audit policies

Audit class	IFCID	Audit category for audit policies	Notes on audit policies
1	140	CHECKING	
2	141	SECMAINT	Also includes IFCID 270, 271.
3	142	OBJMAINT	
4	143	EXECUTE	Records all changes and reads. Can either audit all SQL statements or INSERT, UPDATE, or DELETE only.
5	144		
6	145		
7	55, 83, 87, 169, 319	VALIDATE	Also includes IFCID 269, 312
8	23, 24, 25, 219, 220	CONTEXT	Excludes IFCID 219, 220
9	146		
11	361	DBADMIN SYSADMIN	Can audit specific authorities.

In terms of performance, when using the new audit policies, you need to be aware of which IFCIDs are collected and when, to evaluate the overhead for auditing. With audit policies you can collect *all* the DML accesses to a table. The audit trace instead only records the *first* access within the unit of recovery, which can constitute an incomplete auditing for some institutions. Audit policies also add IFCIDs that were not included with audit traces to some audit categories.

Sample auditing test

Our test uses a simple application which issues 16 separate SELECT statements against a table, we then compare the differences in set up for audit trace and audit policy. Note that the SMF data sets used during the tests have been SWITCHed during the tests but each data set contains more than one test result, so we generated the selected output by using the FROM/TO clause of OMEGAMON PE reporter function when showing the results.

Example 10-1 shows how to start an audit trace using the START TRACE command. In our case, the table we are going to audit is defined with AUDIT ALL in its DDL. If you change the AUDIT specification using the ALTER TABLE statement, the packages related to that table will be invalidated and the auto BIND will take place the next time you execute the application. REBIND the packages explicitly if you have ABIND set to NO, otherwise the execution fails.

Example 10-1 START TRACE for auditing DML access to a table with audit trace

```
-DBOA STA TRACE(AUDIT) CLASS(4,5,6) DEST(SMF)
DSNW130I -DBOA AUDIT TRACE STARTED, ASSIGNED TRACE NUMBER 03
DSN9022I -DBOA DSNWVCM1 '-STA TRACE' NORMAL COMPLETION
```

When you run an OMEGAMON PE reporter to report the trace after a single execution of the application, you get the IFCID frequency distribution log as shown in Example 10-2. In this report you can see the number of IFCID records counted within the input SMF data sets and also the number of records processed for the report. For our static SQL application accessing the table, one IFCID 144 record was processed, or recorded, with the audit trace report, even though 16 SELECTs have been issued. This is due to the limitation of the audit trace, which only records the first access to the table within the unit of recovery.

Example 10-2 OMEGAMON IFCID frequency distribution log for audit trace with static SQL statements

LOCATION: DBOA GROUP: N/P MEMBER: N/P SUBSYSTEM: DBOA DB2 VERSION: V10				OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1) IFCID FREQUENCY DISTRIBUTION LOG				PAGE: 1 RUN DATE: 11-03-10 17:06:51.97		
0	INPUT IFCID	INPUT COUNT	INPUT PCT OF TOTAL	PROCESSED COUNT	PROCESSED PCT OF TOTAL	IFCID	INPUT COUNT	INPUT PCT OF TOTAL	PROCESSED COUNT	PROCESSED PCT OF TOTAL
	1	43	14.19%	0	0.00%	144	23	7.59%	1	2.17%
2	43	14.19%	0	0.00%	145	6	1.98%	0	0.00%	
3	5	1.65%	1	2.17%	202	43	14.19%	43	93.47%	
4	2	0.66%	0	0.00%	225	43	14.19%	0	0.00%	
5	3	0.99%	0	0.00%	239	5	1.65%	1	2.17%	
105	43	14.19%	0	0.00%	362	1	0.33%	0	0.00%	
106	43	14.19%	0	0.00%						
0	TOTAL INPUT TRACE RECORDS =		303							
0	TOTAL PROCESSED TRACE RECORDS =		46							

We now execute the same application with the audit policy. First we populate the SYSIBM.SYSAUDITPOLICIES to collect the EXECUTE *audit category*; this category collects all DML accesses. See Example 10-3.

Example 10-3 Audit policy creation

```
INSERT INTO SYSIBM.SYSAUDITPOLICIES
(AUDITPOLICYNAME, OBJECTSCHEMA, OBJECTNAME, OBJECTTYPE, EXECUTE)
VALUES('AUDTB1','DB2R6','TABLE1','T','A');
```

We then start the audit policy, as shown in Example 10-4. Unlike the audit trace where you need to REBIND after ALTERing your table with the AUDIT specification, in this case auditing starts after you start the audit policies without the need to REBIND the package.

Example 10-4 START TRACE for auditing DML access to a table with audit policies

```
-DBOA STA TRACE(AUDIT) AUDTPLCY(AUDTBS)
DSNW130I -DBOA AUDIT TRACE STARTED, ASSIGNED TRACE NUMBER 03
DSNW192I -DBOA AUDIT POLICY SUMMARY 778
AUDIT POLICY AUDTBS STARTED
END AUDIT POLICY SUMMARY
DSN9022I -DBOA DSNWVCM1 '-STA TRACE' NORMAL COMPLETION
```

Example 10-5 shows the IFCID frequency distribution log with OMEGAMON PE. You can see 16 IFCID 144 records processed for the report. This confirms that an audit record was written for each SELECT statement accessing a table within the application.

Example 10-5 OMEGAMON IFCID frequency distribution log for audit policies with static SQL statements

LOCATION: DBOA GROUP: N/P MEMBER: N/P SUBSYSTEM: DBOA DB2 VERSION: V10		OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1) IFCID FREQUENCY DISTRIBUTION LOG										PAGE: 1 RUN DATE: 11-03-10 17:04:12.58
0	IFCID	INPUT COUNT	INPUT PCT OF TOTAL	PROCESSED COUNT	PROCESSED PCT OF TOTAL	IFCID	INPUT COUNT	INPUT PCT OF TOTAL	PROCESSED COUNT	PROCESSED PCT OF TOTAL	ACTUAL FROM: 11-03-10 00:38:25.43 TO: 11-03-10 01:21:00.00	
		-----	-----	-----	-----		-----	-----	-----	-----	-----	
	1	43	14.19%	0	0.00%	144	23	7.59%	16	26.22%		
	2	43	14.19%	0	0.00%	145	6	1.98%	0	0.00%		
	3	5	1.65%	1	1.63%	202	43	14.19%	43	70.49%		
	4	2	0.66%	0	0.00%	225	43	14.19%	0	0.00%		
	5	3	0.99%	0	0.00%	239	5	1.65%	1	1.63%		
	105	43	14.19%	0	0.00%	362	1	0.33%	0	0.00%		
	106	43	14.19%	0	0.00%							
0	TOTAL INPUT TRACE RECORDS =			303								
	TOTAL PROCESSED TRACE RECORDS =			61								

It is obvious that you get better auditing capability with audit categories. However, depending on the average of DML statements executed per transaction when accessing the same table within the same unit of recovery, additional audit trace records are created when you start using audit policies instead of the audit trace.

Compressing your audit records

When audit policies are used instead of audit traces, you are expected to get more trace records. This is due to several changes to include additional information.

You can use the new DSNZPARM SMFCOMP to save disk space for SMF data sets by compressing your DB2 trace records, as explained in 3.5, "SMF compression" on page 74.

We show how the audit trace can take advantage of the SMF compress function by setting SMFCOMP to ON and using an application which executes 20,000 simple SELECT statements against a table. The trace generates 20,000 IFCID 144 records. The tool OMEGAMON PE can automatically process compressed SMF records.

If you are using SMF as an input to your own application, instead of OMEGAMON PE, you can use the DB2 provided sample program DSNTSMFD to decompress compressed DB2 trace data. APAR PM27872 has provided DSNTSMFD. This program also gives the percentage of space saved using SMF compression.

Figure 10-1 shows the output from the decompression sample program DSNTSMFD.

We see that 20,032 out of the 20,096 DB2 trace records within SMF are IFCID 144 records. Our test application only shows 39% saving because more than 99% of the DB2 trace records are IFCID 144 records. In general, you are expected to have a better compression ratio because you get better compression ratio with accounting trace, statistics trace, and most other audit trace records.

*** DSNTSMFD *** STARTING	2011/03/11	16:49:35
<hr/>		
Total records read:.....	20370	
Total DB2 records read:.....	20293	
Total DB2 compressed records read:.....	20090	
Total DB2 compressed records decompressed:.....	20090	
Total non-DB2 records read:.....	77	
Aggregate size of all input records:.....	4156941	3M
Aggregate size of all input DB2 records:.....	4131027	3M
Aggregate size of all DB2 compressed records:....	3767659	3M
Aggregate size of all output DB2 records:.....	6587780	6M
Aggregate size of all DB2 expanded records:.....	6224412	5M
Aggregate size of all non-DB2 input records:.....	25914	25K
Percentage saved using compression.....	39%	
Details by DB2 subsystem		
Subsystem ID: DBOA		

Figure 10-1 DSNTSMFD output for audit records - IFCID 144

The IRWW workload

In order to evaluate the impact of audit policy, we executed the IRWW workload where a mix of 7 transactions access 8 tables with different profiles. See 5.1.2, “IRWW workload” on page 127 for information about IRWW.

We defined an audit policy with EXECUTE audit category on all IRWW tables. The audit records were written to SMF.

We compare the results of measurements done with the audit function turned off, to the results of measurements with auditing turned on.

Here is a summary of the measurement environment:

- ▶ IRWW:
 - 7 types of transactions; average 6 SQL statements
 - 8 tables
 - 3 clients concurrently running to access DB2
- ▶ z10 processor; 1 CP

The measured values are listed in Table 10-2.

Table 10-2 Measured values in seconds

Audit policy	CL2 elapsed time	CL2 CPU time
Off	0,006016	0,000777
On	0,006258	0,000838

Figure 10-2 shows the average class 2 elapsed time and CPU time from accounting traces for audit policies on against audit policies off. Our results show an increase of 7.85% for CPU time and an increase of 4% for elapsed time increase as overhead for turning on the audit policy with EXECUTE audit category.

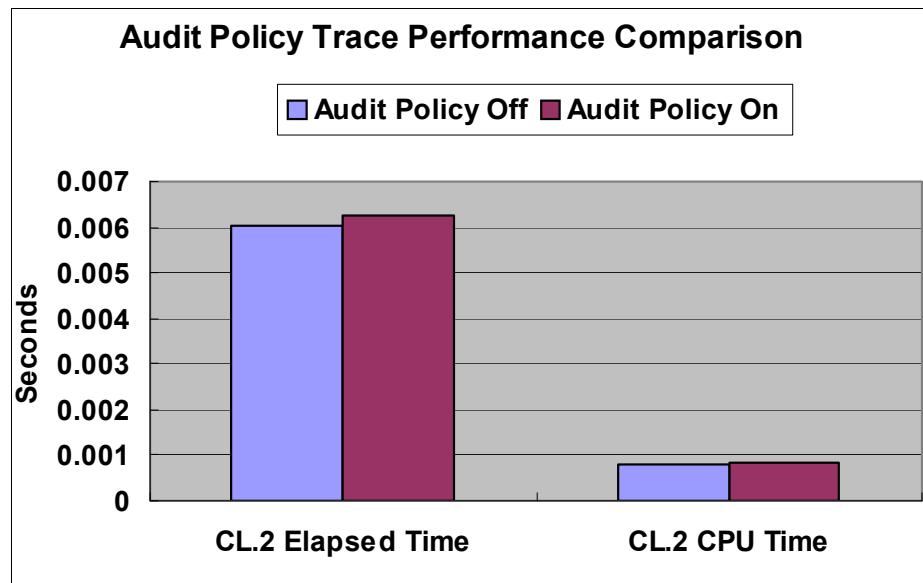


Figure 10-2 Audit policies performance

Another consideration on activating auditing is the amount of space required to store the trace records in SMF data sets, as discussed in “Compressing your audit records” on page 295 regarding compressing the trace records.

Using the trace records taken in this measurement, we compare the size of the audit records with the basic IFCID 3 accounting records. Example 10-6 lists the trace counts from the SMF output without audit policies. The disk storage size is 112 cylinders.

Example 10-6 IFCID counts with audit policies turned off

INPUT	
IFCID	COUNT
3	30,000

After turning on the audit policy and having the same workload executed, the trace record counts is as shown in Example 10-7. The disk storage size is now 201 cylinders.

Example 10-7 IFCID counts with audit policies turned on

INPUT	
IFCID	COUNT
3	30,000
143	71,832
144	114,955

For read and write auditing records in the 6:4 ratio in the DML accessing the tables, our measurement results show an average of about 1/8 in size for the audit record size versus IFCID 3 record size.

10.1.2 Benefits of DB2 10 audit policies

With the audit policies capability added to DB2 10, you get two benefits:

- ▶ You can gather multiple SQL accesses to a table from same unit of recovery where audit trace only records the first access to a table.
- ▶ You can gather audit traces without ALTERing your table with AUDIT parameter (which invalidates the packages).

Our performance results show a reasonable overhead against the IRWW workload where you can expect around 7-8% CPU time overhead for the defined profile, averaging 6 SQL statements per transaction. Overhead will vary with the type and the number of SQL statements. More complex queries will see less % overhead. More statements will create more trace records and increase the overhead.

During our tests, we observed that the audit policy writes trace records differently depending on how you code your application when repeating the execution of the same SQL statement. For example, a 10 time loop of a single SQL statement such as the one shown in Example 10-8 gathered only 1 IFCID 144 record when the host variable :ICODE is being changed in a loop to retrieve 10 records from a table. DB2 10 treats repetition of the same SQL statements accessing a table as a cursor access and you will get only 1 IFCID 144 record during a execution.

Example 10-8 A SQL statement sample in loop

```
...(loop)
EXEC SQL SELECT * INTO :DCLTABLE1
    FROM DB2R6.TABLE1
    WHERE T_A_CODE      = :ICODE
END-EXEC
...
```

If you create an application to get the same results as the previous application but with two separate SQL statement in a 5 time loop, as shown in Example 10-9, each SQL statement in the loop has a different value and DB2 gives you 10 IFCID 144 records.

Example 10-9 Two SQL statements in loop

```
...(loop)
EXEC SQL SELECT * INTO :DCLTABLE1
    FROM DB2R6.TABLE1
    WHERE T_A_CODE      = :ICODE
END-EXEC
...(logic)
EXEC SQL SELECT * INTO :DCLTABLE1
    FROM DB2R6.TABLE1
    WHERE T_A_CODE      = :ICODE2
END-EXEC
...
```

You can expect such results when you get a loop of an SQL statement with a unique STMT_ID. Even in the case of exactly the same SQL statement, such as the second SQL statement in Example 10-9 where the SQL statement has :ICODE instead of :ICODE2, DB2 will assign to each SQL statement a separate STMT_ID and therefore you get multiple IFCID 144 records.

If you are running dynamic SQL statements with dynamic statement cache, executing the same SQL statement in the loop will result in 1 IFCID 144 because a single STMT_ID is assigned and cached. You will not have different STMT_ID as static SQL applications do.

10.2 Support for row and column access control

DB2 10 introduces a method for implementing row and column access control as an additional layer of security that you can use to complement the privileges model and to enable DB2 to take part in your efforts to comply with government regulations for security and privacy. You use row and column access control to control SQL access to your tables at the row level, column level, or both, based on your security policy.

To implement row and column access control, DB2 10 added new objects to filter rows and columns or data:

- ▶ Row permission:
 - SELECT, INSERT, UPDATE, DELETE, MERGE
 - Separates security logic from application logic
 - Determines which rows are returned
- ▶ Column mask:
 - SELECT, INSERT, UPDATE, MERGE
 - Determines how the columns in each row are returned

After you enforce row-level or column-level access control for a table, any DML statement that attempts to access that table is subject to the row and column access control rules that you define. During table access, DB2 transparently applies these rules to every user, including the table owner and the install SYSADM, SYSADM, SECADM, system DBADM, and DBADM authorities, which can help to close security loopholes.

Currently customers can use views to hide column and row values, however, DBAs and application developers need to define, use, and maintain them to have users securely access the data. Furthermore, views might encounter an issue when updates are executed or auditing is needed and might require triggers.

10.2.1 Row permission performance

We evaluated the impact of having row permission, where internal query rewrite can occur. The measurement was done using the IRWW workload defining row permissions set on all 9 IRWW tables. See “IRWW OLTP performance” on page 127 for information about IRWW.

Example 10-10 shows one of the row permission definitions used for the measurement. To see the impact of having row permission, the row permissions defined do not filter out any rows from table in order to ensure the comparability with the baseline test. In real life the definition might cause DB2 to choose a different access path.

Example 10-10 Sample row permission for measurement

```
CREATE PERMISSION RP1 ON USRT001.WAREHOUSE FOR ROWS
WHERE W_ID <> '0000'
ENFORCED FOR ALL ACCESS
ENABLE;
ALTER TABLE USRT001.WAREHOUSE ACTIVATE ROW ACCESS CONTROL;
```

The measurement environment is as follows:

- ▶ JCC level 2.10.72
- ▶ 3 CP
- ▶ Application:
 - 10 concurrent clients
 - No think time

Figure 10-3 shows the ITR results from IRWW workloads with and without row permissions. We observed a runtime throughput regression of 2.1%. You will have some more overhead during the prepare processing, but overhead is minimum for runtime.

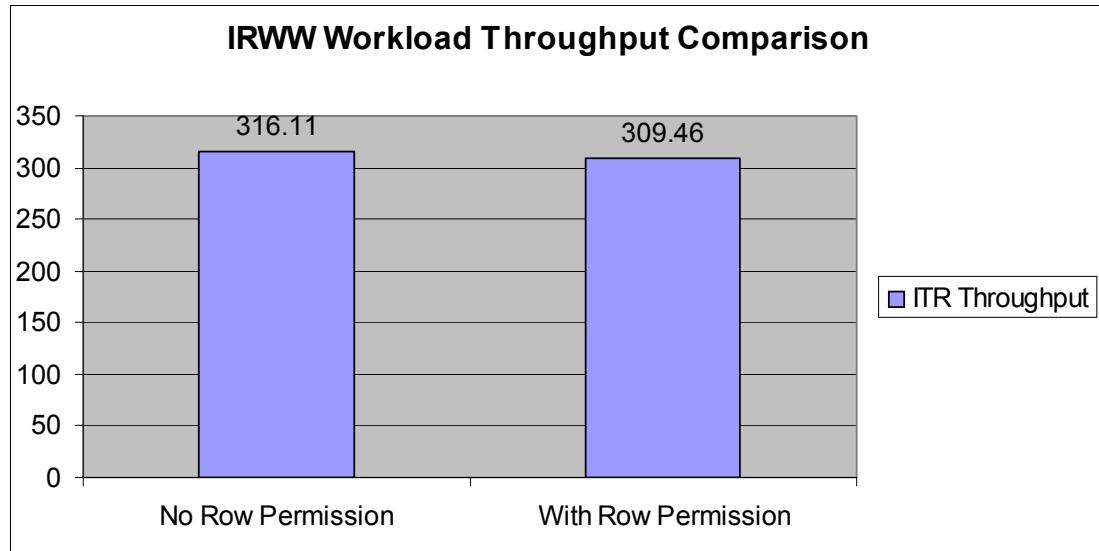


Figure 10-3 ITR with row permissions

10.2.2 Column mask performance

The new row-level or column-level access control for a table is similar to having a view defined on a table. We have done some measurements using the column mask comparing them to an equivalent view defined on the table.

We define a column mask on the table where part of the data is masked from a table for all users except when the user has role ROLE1 defined. See Example 10-11.

Example 10-11 Column mask definition

```
CREATE MASK M1 ON ADMF001.EMP_E X1
  FOR COLUMN SSN RETURN CASE
    WHEN(
      VERIFY_ROLE_FOR_USER(SESSION_USER, 'ROLE1')=1
      ) THEN SSN
    ELSE 'XXX-XX-' || SUBSTR(X1.SSN,8,4)
  END
ENABLE;
```

Example 10-12 shows the view definition equivalent to the column mask definition listed in Example 10-11.

Example 10-12 View definition

```
CREATE VIEW ADMF001.EMP_E_VIEW(C1, C2, C3, C4, C5) AS
SELECT ID,
CASE
WHEN(
    VERIFY_ROLE_FOR_USER(SESSION_USER, 'ROLE1')=1
) THEN SSN
ELSE 'XXX-XX-' || SUBSTR(SSN,8,4) END SSN
, BDATE, SALARY, BONUS FROM ADMF001.EMP_E;
```

The measurements of two SELECTs against a table with column mask show less than 1% runtime CPU difference when compared with running against an equivalent view. The results can be considered equivalent, but you will find the column mask overall more beneficial because DBAs or application developers do not have to worry about managing views to control access to data.

The queries against row permission protected tables get rewritten with predicates defined by row permission. You can expect to have up to 35% class 2 CPU overhead for query rewrite at BIND or full prepare time. But our measurement results show minimum impact on runtime using existing access paths.

Consider that the access path will be chosen after query rewrite. So, if you have the row permission which adds stage 2 predicates or subquery predicates, your overall query performance can degrade. Vice versa, you might be able to improve your overall performance by adding screening predicates or stage 1 predicate with row permission.

The statement sharing in dynamic statement cache is not affected by row permissions because you will start getting the same queries after the rewrites.

10.3 Recent maintenance notes

APARs PM26977 and PM28296 provide enhancements to the new DB2 authorities.

PM26977 separates system privileges from system DBADM authority. System DBADM will no longer have implicit system privileges, STOPALL, USE of BUFFERPOOL, USE of STOGROUP and will not have implicit privilege to issue system commands such as STOP DB2, START/STOP/MODIFY DDF, START/STOP RLIMIT, CANCEL THREAD/DDF THREAD and RESET GENERICLU.

PM28296 adds support for secure audit policy trace.

A new value 'S' is added to SYSIBM.SYSAUDITPOLICIES - DB2START column. If 'S' is specified, then the audit policy will be automatically started at DB2 start up and can only be stopped by user with SECADM authority or will be stopped at DB2 stop.

- ▶ If multiple audit policies are specified to be started at DB2 start up, some with DB2START = 'Y' and some with DB2START = 'S', then two traces will be started, one for audit policies with DB2START='Y' and another for audit policies with DB2START='S'. To stop the audit policies started at DB2 start up, all the audit policies that are assigned the same trace number must be stopped simultaneously. Then the policies can be started individually, as needed.

- ▶ After adding the entry in SYSAUDITPOLICIES with DB2START='S', the trace can be started to get this behavior.
- ▶ Any user with the necessary privilege can start the trace specified with DB2START = 'S' and the SECADM restriction applies only during stop trace.

If the audit policy has already been started, then after updating the DB2START column, the audit policy needs to be stopped and restarted to get the secure behavior.



Installation and migration

DB2 10 for z/OS has modified the structure of the catalog and added several new objects for the support of new functions. Furthermore, steps have been added to the installation procedures to help users in the set up of specific environments such as SAP and DB2-supplied stored procedures.

In this chapter we describe the installation requirements and the performance implications of moving to CM and NFM from both DB2 9 and DB2 V8.

We discuss the following topics:

- ▶ Before you start
- ▶ Installation
- ▶ Migration

11.1 Before you start

Before you begin the installation or migration process, look at the big picture. Be aware of the major requirements to get from your current version of DB2 to DB2 10 for z/OS. You need to know where you are currently and where you need to be before you embark on this process considering DB2, z/OS, and tools. Decide if it is convenient for you to take advantage of skip-level migration from DB2 V8 NFM directly to DB2 10 conversion mode (CM). After you have started a migration, you are able to fall back, but you can only re-migrate the same way you tried before because the catalog will have the imprint of the level you went to.

Figure 11-1 shows the life cycle of the versions of DB2 and the minimum level of z/OS that is required. At the time of writing this book, the latest three versions of DB2 are fully supported, and end of service for DB2 V8 is scheduled for 30 April 2012.

If you are running DB2 V7, you can order DB2 V8, although it is withdrawn from marketing.

Version	PID	Generally available	OS prereq	Marketing withdrawal	End of service
V3	5685-DB2	December 1993	MVS V4R3	February 2000	January 2001
V4	5695-DB2	November 1995	MVS V4R3	December 2000	December 2001
V5	5655-DB2	June 1997	MVS V4R3	December 2001	December 2002
V6	5645-DB2	June 1999	OS/390 V1R3	June 2002	June 2005
V7	5675-DB2	March 2001	OS/390 V2R7	March 2007	June 2008
V8	5625-DB2	March 2004	z/OS V1R3	September 2009	April 2012
V9	5635-DB2	March 2007	z/OS V1R7	December 2012	June 2014
V10	5605-DB2	October 2010	z/OS V1R10	TBD	TBD

Figure 11-1 DB2 version summary

Consider also the impact of the “right” level of z/OS. The minimum prerequisite is z/OS 1.10, but some functions might require a later version.

11.2 Installation

The installation process is documented in more detail in two publications. See *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974, and Chapter 12.” Installation and migration” of *DB2 10 for z/OS Technical Overview*, SG24-7892.

DB2 10 for z/OS, program number 5605-DB2, consists of following function modification identifiers (FMIDs):

- ▶ Required FMIDs:
 - HDBAA10 (contains DB2 Base, REXX, IBM MQSeries®, MQListener)
 - HIYAA10 (IMS Attach - must be installed even if you do not have IMS)
 - HIZAA10 (Subsystem Initialization)

- HIR2230 (IRLM V02.03.00)
- HDREA10 (DB2 RACF® Authorization Exit)
- JDBAA14 (DB2 English Panels)
- ▶ Optional FMIDs:
 - JDBAA12 (DB2 JDBC/SQLJ)
 - JDBAA17 (DB2 ODBC)
 - JDBAA11 (DB2 Kanji Panels, available at General Availability)

This information extracted from the *Program Directory for DB2 10 for z/OS*, GI10-8829. See the latest versions of such publications for any changes to the DB2 10 for z/OS prerequisites that might affect your installation.

Version 10 of DB2 Utilities Suite for z/OS, program number 5655-V41, is comprised of FMID JDBAA1K and contains DB2 utilities that are not provided in the DB2 base.

If you are to use other DB2 Tools with DB2 10, be sure you have the appropriate release. You can find information about compatibility with DB2 10 at the following website:

<http://www.ibm.com/support/docview.wss?uid=swg21409518>

There are also orderable no-charge and charge features as well as separate downloadable features that can work with DB2 10 for z/OS. See *DB2 10 for z/OS Technical Overview*, SG24-7892 for more information.

11.3 Migration

Migration is the process of upgrading from a DB2 9 environment to DB2 10 environment. DB2 also supports a *skip level migration* where you are upgrading a DB2 V8 environment to a DB2 10 environment. In this section, we assume that the DB2 product being migrated is installed in new-function mode.

11.3.1 Introduction to migration to DB2 10

You can migrate to DB2 10 either from DB2 9 or DB2 V8 in new-function mode. DB2 10 has a minimum requirement for z/OS to be at least at level V1.10 and at V1.11, if you want to remove some RECOVER utility restrictions for object recovery from system-level backup and Extended Address Volumes (EAV) for large sequential data sets. DB2 10 takes advantage of a new z/OS 1.12 interface to enable data sets to be opened more quickly and to maintain more open data sets.

The DB2 installation might require a migration to new z/OS releases to be service supported. Take that into account when planning for migration.

DB2 10 operates on any processor that supports 64-bit mode, including zEnterprise 196 (z196), z10, z9, z990, z890, and later processors that are supported by z/OS 1.10. See the *Program Directory for DB2 10 for z/OS*, GI10-8829, for more information about system requirements.

During the migration to DB2 10, the system programmer must be aware of how certain system parameters impact performance. Often the system parameters are chosen based on defaults and are carried along, but sometimes they are changed across releases to reflect changes in hardware and software. You might want to adjust them back when you migrate to compatibility mode and want to compare system behavior.

The migration path to DB2 10 continues to use the same general approach as with DB2 9 and DB2 V8. Because DB2 10 supports skip level migration, DB2 migration modes are renamed to take into account the DB2 release which you are migrating from and the release you fallback to. The two graphs in Figure 11-2 summarize the migration paths and the resulting migration modes.

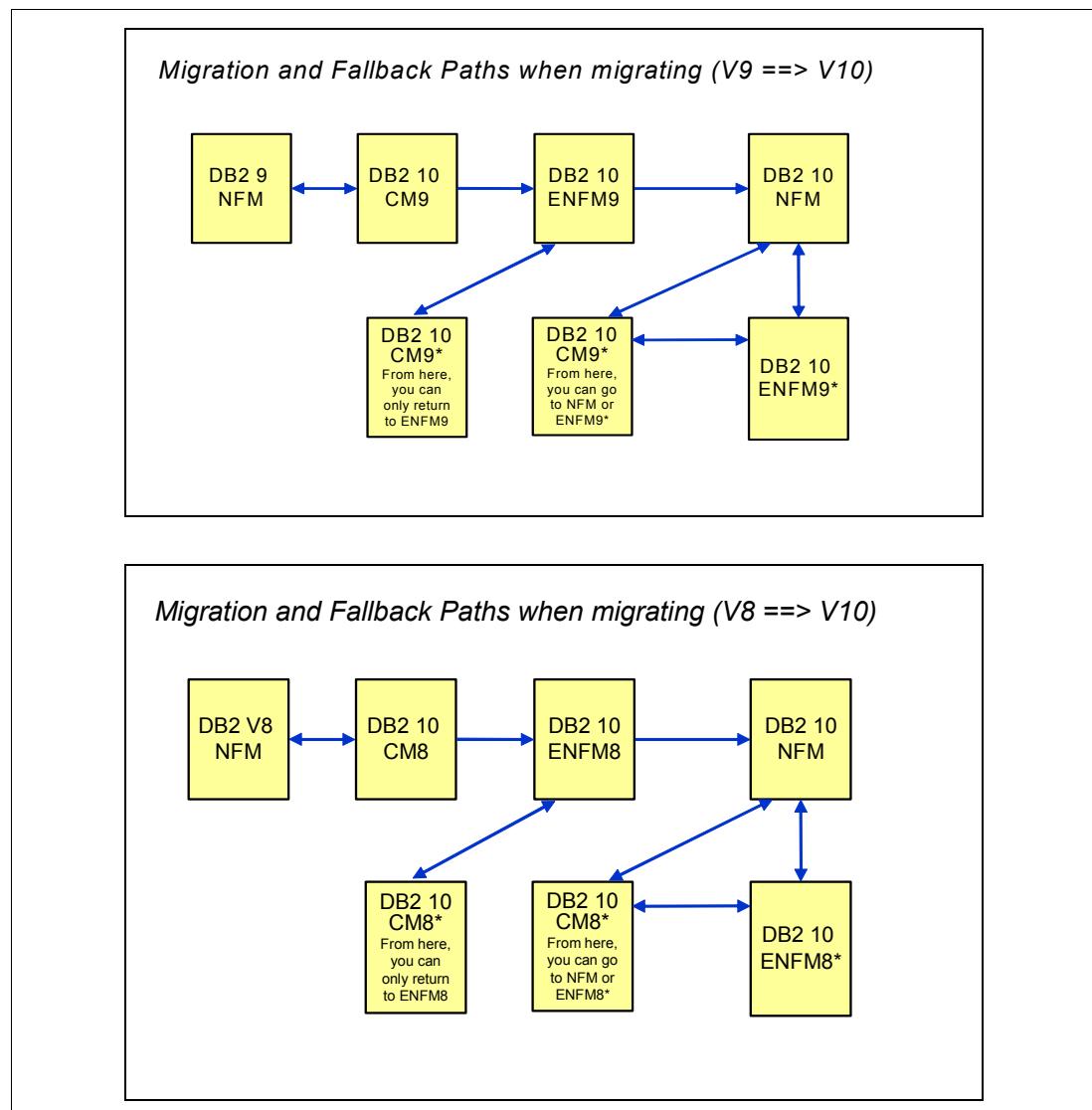


Figure 11-2 Migration paths and modes

DB2 supports the following modes:

- ▶ Conversion mode 9 (CM9) or conversion mode 8 (CM8):

This is the DB2 mode where DB2 10 has migrated from the DB2 9 or DB2 V8 environment for the first time. It is still in conversion mode when the migration job DSNTIJTC that has been run to perform the CATMAINT function has completed. DB2 can only migrate to conversion mode from new-function mode. Some new functions are available in conversion mode.

For data sharing system, coexistence with a prior release of DB2 new-function mode system is possible, briefly, such as over a weekend or a week. We try to move most (but not all) problems for the migration from new-function mode and ENFM to conversion mode, so that fallback to the prior DB2 system can be used, if necessary.

- ▶ Enabling new-function mode 9 (ENFM9) or enabling new-function mode 8 (ENFM8)
 This mode is entered when CATENFM START is executed, which is the first step of migration job DSNTIJEN. DB2 remains in this mode until all the enabling functions are completed. Note that all the members in a data sharing group need to be migrated to DB2 10 conversion mode because you cannot mix DB2 9 or DB2 V8 new-function mode and DB2 10 ENFM in a data sharing group.
- ▶ New-function mode (NFM)
 This mode is entered when CATENFM COMPLETE is executed, using migration job DSNTIJNF. This mode indicates that all catalog changes are complete and all DB2 10 new functions can be used.
- ▶ ENFM9* or ENFM8*
 This mode is same as ENFM9 or ENFM8, but the asterisk (*) indicates that DB2 was migrated to NFM but fallen back to ENFM9 or ENFM8, whichever mode you had migrated from. Objects that were created when the system was at NFM can still be accessed, but no new objects can be created. When the system is in ENFM9*, it cannot fall back to DB2 9 or coexist with the DB2 9 system. The same rules apply to ENFM8* and a DB2 V8 system.
- ▶ CM9* or ENFM8*
 This mode is the same as CM9 or CM8, but the asterisk (*) indicates that DB2 has migrated to a higher level of modes. Objects that were created at the higher level can still be accessed. When DB2 is in CM9* mode, it cannot fall back to DB2 9 or coexist with a DB2 9 member for a data sharing group. The same rules apply to CM8* and DB2 V8.

The following fallback is possible during migration:

- ▶ Fallback from CM9 to DB2 9 system
- ▶ Fallback from CM8 to DB2 V8 system
- ▶ Fallback from ENFM9 to CM9*
- ▶ Fallback from ENFM8 to CM8*
- ▶ Fallback from NFM to ENFM9*, ENFM8*, CM9*, or CM8*.

It is only possible to fallback to the mode you have migrated from. So if you have migrated to DB2 10 from DB2 9, you can fallback to either ENFM9* or CM9* but not to ENFM8* nor CM8*.

In conversion mode, no new DB2 10 function is available for use that might preclude the fallback to DB2 9 or DB2 V8. To use some functions, the DB2 subsystem or data sharing group must first convert the DB2 catalog to a new-function mode catalog by the ENFM process.

Skip level migration

This capability delivers greater flexibility as to how and when you can migrate to DB2 10. However, a skip level migration requires careful planning to make the correct decisions. Be aware that preparation for a skip level migration does require more work than a migration from DB2 9, because you need to consider incompatibilities for DB2 9 as well as DB2 10. The actual skip level migration process itself is rather simple and allows you to jump directly from DB2 V8 NFM to DB2 10 CM8 and to DB2 10 NFM without any intermediate steps of going through DB2 9.

Premigration work summary

To prepare for migration to DB2 10, the following list includes items that you must consider before migrating to DB2 10 conversion mode:

- ▶ DB2 provides you with a premigration job DSNTIJPA in the DB2 release you are migrating from. This job is delivered by the service stream in APAR PM04968. You can run this job at any time during the premigration planning stages to determine the cleanup work that you need to do to DB2 9 or DB2 V8 to prepare for migration to DB2 10. You also need to apply the fallback SPE APAR PK56922 and prerequisite fixes. Read the HOLD data for APAR PK56922. See also information APARs II14474 and II14477. See *DB2 10 for z/OS Technical Overview*, SG24-7892, for more information.
- ▶ For old plans and packages from DB2 V5 or earlier version, you need to REBIND them, or DB2 automatically rebinds them if you set DSNZPARM ABIND=YES. You also need to convert plans containing DBRMs to packages prior to migrating; the functionality for converting plans is available in DB2 9 and DB2 V8 by APARs.
- ▶ DB2 10 requires the use of partitioned data set extended (PDSE). You must define SDSNLOAD and SDSNLOD2 libraries as a PDSE instead of a partitioned data set (PDS).
- ▶ If you are migrating to DB2 10 from DB2 V8, make sure that the bootstrap data sets (BSDS) are converted to the expanded format that became available in DB2 V8. This expanded format supports up to 10,000 entries per copy of archive logs and up to 93 entries per copy of active logs, and entries for TCP/IP v6.

See *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974, and Chapter 12, “Installation and migration” of *DB2 10 for z/OS Technical Overview*, SG24-7892 for more information.

The migration step to CM

Migration to CM9 or CM8 is achieved by running the CATMAINT UPDATE migration job DSNTIJTC, which does the following functions:

- ▶ Adds new catalog table spaces and tables
- ▶ Adds new columns to existing catalog tables
- ▶ Adds new meanings to existing catalog columns
- ▶ Adds new indexes for new and existing catalog tables

The migration step to ENFM

Migrating to ENFM9 or ENFM8 is achieved by running the migration job DSNTIJEN, which does the following functions by online REORG:

- ▶ Adds new table spaces replacing old table spaces
- ▶ Adds new tables and indexes
- ▶ Adds new columns to existing catalogs
- ▶ Adds new LOB table spaces, auxiliary tables, and auxiliary indexes

The migration step to NFM

Migration to NFM is achieved by running the migration job DSNTIJNF, which enables new functions by executing CATENFM COMPLETE.

11.3.2 Summary of catalog changes

DB2 10 has made restructure of DB2 catalog to ease the limitation related to catalog and directory. The following lists summarize benefits by the change. See *DB2 10 for z/OS: Technical Overview*, SG24-7892 for details.

- ▶ Eases the physical and management limitation:
 - Bypass the 64 GB limit for catalog and directory table spaces
 - DB2 catalog tables in a partition-by-growth table space
 - SMS managed data sets for the catalog and directory
- ▶ Provides more concurrency:
 - Catalog contention reduction for removal of links and introduction of row level locking
 - Multiple binds running concurrently
 - Less contention for automatic REBIND on invalid packages
 - Updates the statistics and catalog pages without getting a deadlock as frequently
 - Updates the referential integrity information and catalog pages without getting a deadlock as frequently

If your applications do not use LOBs today and if you are not accustomed to LOBs, be aware that the DB2 10 catalog now has LOB objects. This implies catalog recovery procedure changes and related disaster recovery testing.

See also 2.1, “Catalog restructure” on page 16.

11.3.3 Catalog migration

As we have seen, there are two steps involved in the catalog migration to DB2 10, as there were for migration to DB2 9 and DB2 V8:

1. Migrate to conversion mode:

As a part of the migration to DB2 10 conversion mode (either CM8 or CM9), you run supplied and tailored job DSNTIJTC. This job runs the DSNUUTILB utility program with the parameters CATMAINT UPDATE. As for previous release of DB2 does, CATMAINT has been part of the standard DB2 migration process.

2. Enable new-function mode:

The job you run is DSNTIJEN. It starts with CATENFM START to enter ENFM, and the CATENFM CONVERT and online REORG to the catalog table spaces which change them into new structures.

Then you run DSNTIJNF, the job runs the DSNUUTILB utility program with the parameter CATENFM COMPLETE which will switch DB2 10 to NFM.

You can verify the status by entering the DISPLAY GROUP command (Example 11-1).

Example 11-1 Sample output from Display Group command

```
-DB0A DIS GROUP
DSN7100I  -DB0A DSN7GCMD 658
*** BEGIN DISPLAY OF GROUP(.....) CATALOG LEVEL(101) MODE(NFM )
      PROTOCOL LEVEL(3) GROUP ATTACH NAME(....)

-----  

DB2                               DB2 SYSTEM     IRLM
MEMBER   ID  SUBSYS  CMDPREF    STATUS   LVL NAME      SUBSYS  IRLMPROC
-----  

.....   0  DBOA   -DB0A      ACTIVE   101 SC63      ID0A    DBOAIRLM

-----  

*** END DISPLAY OF GROUP(.....)
DSN9022I  -DB0A DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
```

11.3.4 Rebind during migration

REBIND is not required for migration to DB2 10, but consider REBIND during migration to take advantage of DB2 10. Getting the best performance improvements and eliminating regression does depend upon REBIND in most situations (for example getting current structures, better access paths, and reusing threads).

Improvements in access paths can be significant, such as stage 2 predicates that can become stage 1. REBIND processing in DB2 10 takes more CPU and elapsed time than in prior versions, but more concurrent REBINDs are possible when you are in NFM.

Starting from DB2 9, DB2 for z/OS has introduced new options for REBIND PACKAGE called an access plan stability which can help in case of regression. When you migrate to DB2 10, take advantage of the function to use an access plan stability because it provides, in case of regression, an easy way to restore a previous package with previous access path with a REBIND SWITCH command. Be sure that you have sufficient space left for your DB2 directory SPT01, because access stability capability will keep your old package within the DB2 directory SPT01. The default value to use the access plan stability, DSNZPARM PLANMGMT, has been changed to EXTENDED, which will keep two previous copies of the package when possible by default.

11.3.5 Migration steps and performance

This section provides information that relates to the utility performance related to the migration of the DB2 catalog and directory from DB2 9 to DB2 10 conversion mode and DB2 10 enabling new-function mode. The measurements results are from a case study with three different customer scenarios for the timings of the migration processes. The case study is the same that was used in *DB2 9 for z/OS Performance Topics*, SG24-7473, and *DB2 UDB for z/OS Version 8 Performance Topics*, SG24-6465.

Measurements: Although the same catalog and directory was used for migration measurements, you might see different measurement results from previous books because of measurement environment differences.

The measurement environment for non-data sharing includes the following items:

- ▶ DB2 for z/OS V8, DB2 9, and DB2 10
- ▶ z/OS 1.11
- ▶ DS8300 DASDs
- ▶ 4-way (IBM System z10® processor)
- ▶ BP0 20,000, BP4(work) 10,000, BP8K0 10,000, BP16K 10,000, BP32K 50,000

The measurements were done using three company catalog scenarios with three different catalog sizes, as follows:

- ▶ Company 1 (C1) catalog size 28.3 GB
- ▶ Company 2 (C2) catalog size 15.2 GB
- ▶ Company 3 (C3) catalog size 0.698 GB

Unless noted, all the measurements are done using the environment and settings given.

We show the performance results related to the following steps:

- ▶ Migration to DB2 10 CM9
- ▶ Migration to DB2 10 ENFM9
- ▶ Skip level migration
- ▶ Data sharing measurements

Migration to DB2 10 CM9

During the migration from DB2 9 new-function mode to DB2 10 conversion mode 10 (CM9), the job DSNTIJTC uses the CATMAINT utility to update the catalog.

DSNTIJTC execution for the three catalogs

We show the measurement results from running the CATMAINT job on the three catalog sizes. The objective is to have some idea of the duration of the execution (unavailability) and how the catalog sizes might impact the CATMAINT job elapsed and CPU time. See Figure 11-3.

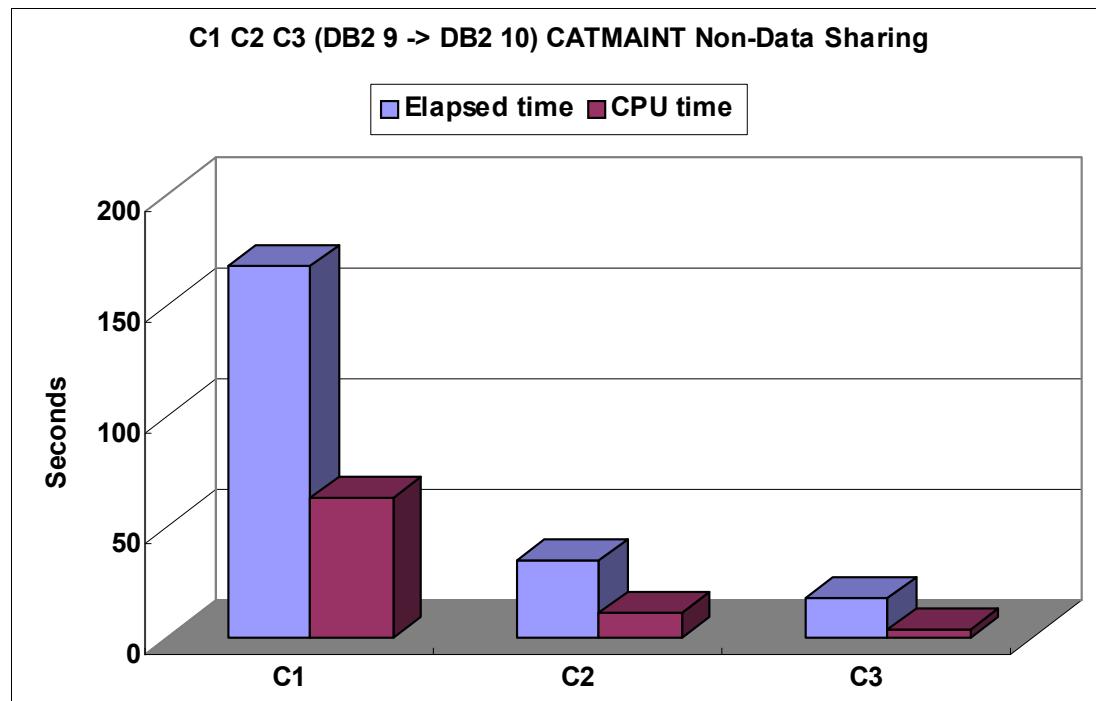


Figure 11-3 CATMAINT elapsed and CPU time comparison

The results might help to predict the performance and elapsed times for the DSNTIJEN job on your system.

This assumes that your catalog has no major anomalies and is not larger than 30 GB. Take into account that the size of the catalog is just one of many factors that can impact performance of the execution time. Even for similar catalog sizes as those shown, results can vary, depending on the number of catalog records, the average package size, as well as several other factors that might affect the processing time.

Figure 11-4 shows the same results, including the catalog size.

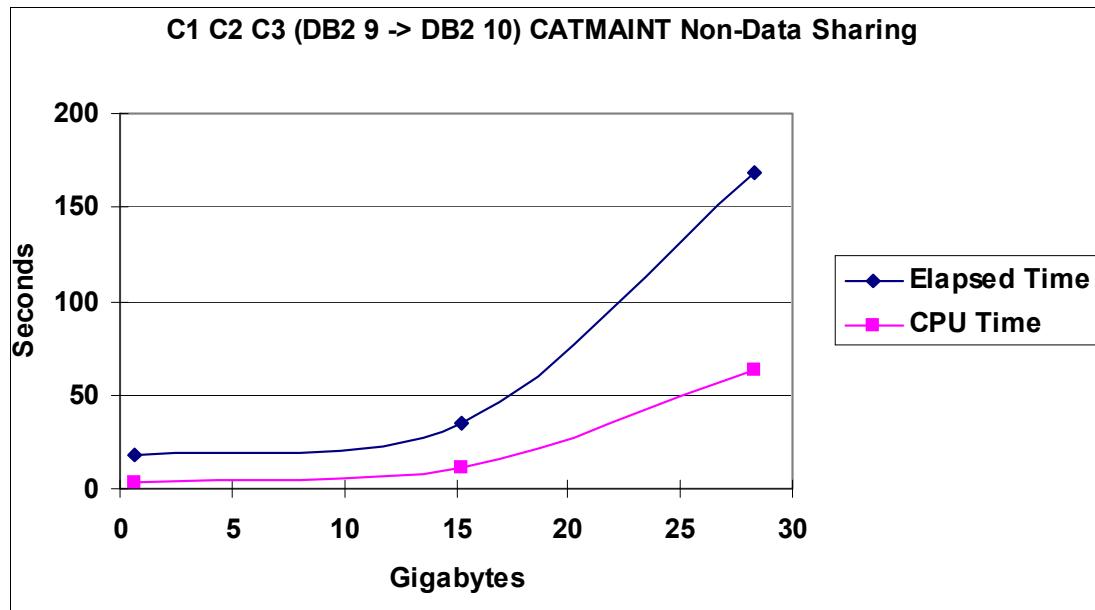


Figure 11-4 CATMAINT execution for the three catalogs

Comparing with previous migrations

Figure 11-5 shows the result of CATMAINT job elapsed time and CPU time using the largest C1 catalog to compare CATMAINT elapsed and CPU time with previous release migrations. The measurements were done migrating the DB2 V7 catalog to the DB2 V8 catalog, then the DB2 V8 catalog to the DB2 9 catalog, and finally the DB2 9 catalog to the DB2 10 catalog reporting execution times. The measurements were done specifically to compare the results of the CATMAINT job, using the same catalog. The results will vary if you make changes or add objects after each migration of the DB2 catalog.

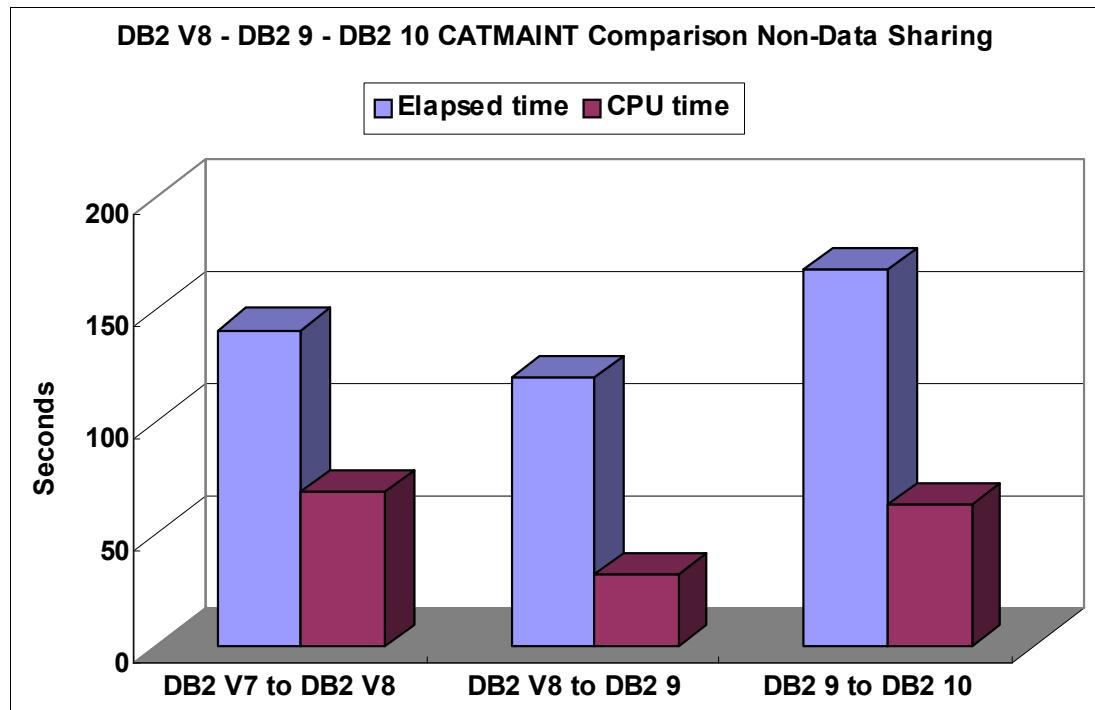


Figure 11-5 CATMAINT elapsed and CPU time comparing with previous releases

We observe that the amount of CPU time needed for CATMAINT from DB2 9 to DB2 10 is similar to that of DB2 V7 to DB2 V8. An elapsed time increase is observed when comparing the migration to DB2 10 to both other migrations, to DB2 V8 and to DB2 9.

Buffer pool tuning for CATMAINT job

During the migration to DB2 10 CM9, we also observed 40% more getpages in BP0, 4 times more getpages in BP8K0 and 40% more getpages in BP32K from our CAMAINT measurement compared with the previous CATMAINT job.

We also did a set of measurements with larger buffer pool sizes for BP0 and BP8K0 to see the effects to the CATMAINT job using the C1 catalog. The results did not show significant differences when increasing the sizes of the buffer pools.

Migration to DB2 10 ENFM9

The CATENFM utility enables a DB2 subsystem to enter DB2 10 ENFM9 and online REORG to convert table spaces into new format then to DB2 10 NFM.

CATENFM execution for the three catalogs

The first measurement compares enable new-function mode job, DSNTIJEN, with elapsed and CPU time using company catalog scenarios with different catalog sizes. See Figure 11-6.

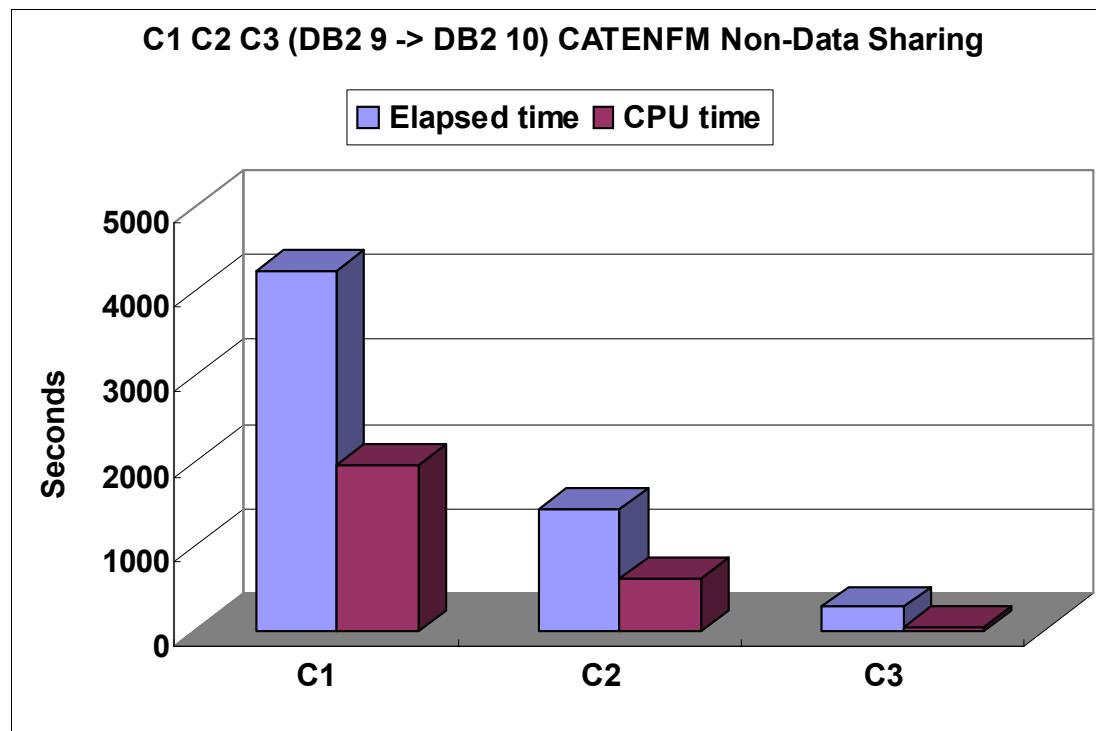


Figure 11-6 CATENFM elapsed and CPU time comparison

The results give an idea of how the catalog size will affect the elapsed and CPU time for DSNTIJEN job. The results shows that both elapsed time and CPU time increase as the catalog size increases, similarly to the CATMAINT job.

Figure 11-7 shows the same results including the catalog size.

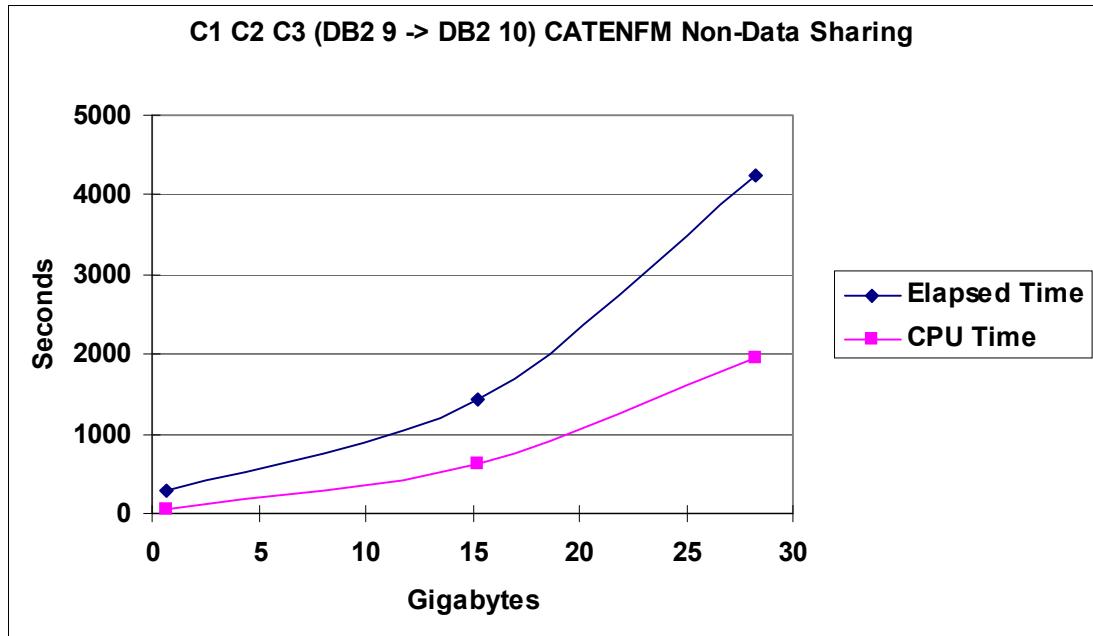


Figure 11-7 CATENFM execution for the three catalogs

Comparing with previous migrations

Figure 11-8 shows the enabling new-function mode jobs, comparing elapsed and CPU time with previous releases. All measurements are done using the initial measurement settings.

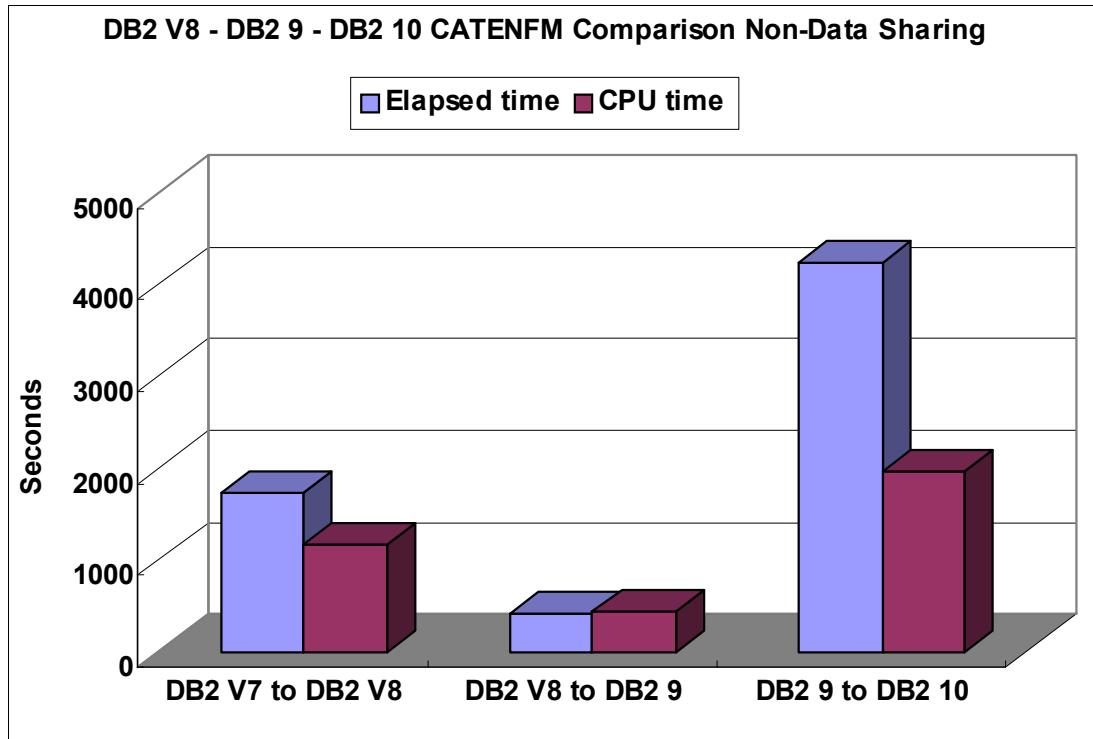


Figure 11-8 The DSNTIJEN elapsed and CPU time comparison with previous releases

The results give an idea of what to expect from the DSNTIJEN job for migrating to DB2 10 NFM. DB2 10 migration results in about 1.7 times more in CPU time and about 2.5 times in elapsed time when compared with the DB2 V8 DSNTIJEN job. This is due to the restructuring of the DB2 catalog as described in 2.1, “Catalog restructure” on page 16.

Most of the time taken within the DNSTIJEN job comes from the online REORG step where 9 table spaces are converted to new structures. When we examine each step, it shows that the largest table spaces are most time consuming within the job.

DSNTIJEN processes 9 table spaces with online REORG. The detailed measurement only show the top 3 time consuming table spaces (SPT01, SYSDBASE, SYSPKAGE). The results shown are for the C1 catalog.

Figure 11-9 gives a breakdown of elapsed time and CPU time consumed within the DSNTIJEN job measurement.

The DB2 directory SPT01, where the packages are stored, is the largest table space and the most time consuming table space in our result.

The second largest is the DB2 catalog SYSPKAGE, where package definitions are held. The number of records is the largest of the three data sets.

The third one is the DB2 catalog SYSDBASE, where your databases, table spaces, tables definitions are stored.

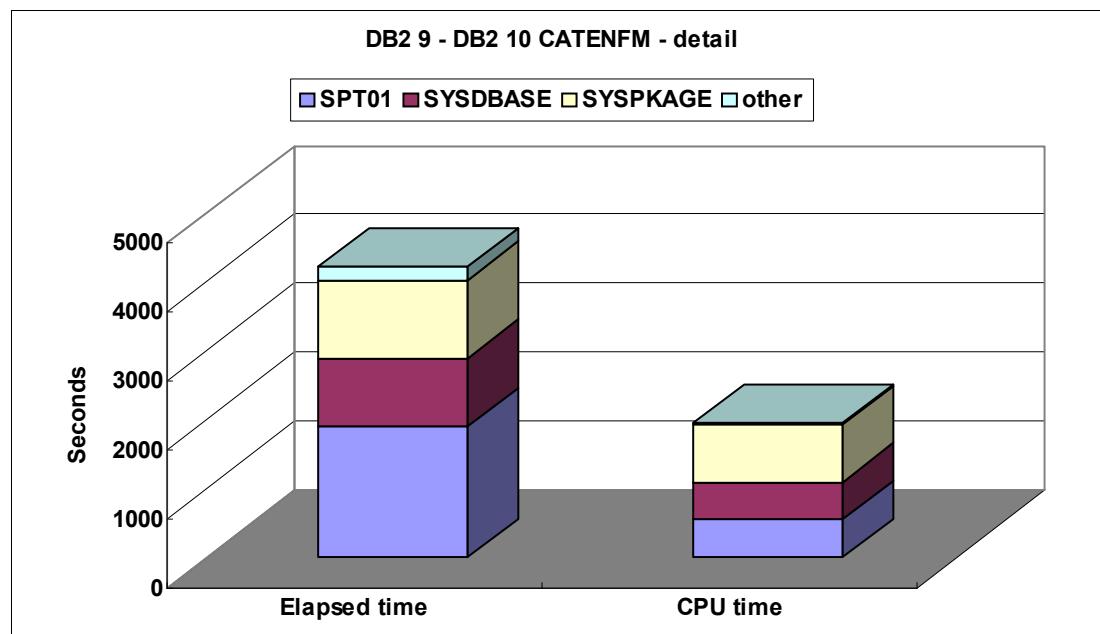


Figure 11-9 DSNTIJEN measurement details

The DB2 catalog SYSPKAGE and the DB2 catalog SYSDBASE are the most CPU intensive and the DB2 directory SPT01 is the most I/O intensive due to a large synchronous I/O wait.

Buffer pool tuning for DSNTIJEN job

Similarly to CATMAINT, the DSNTIJEN job issues more getpage requests to migrate to DB2 10. Unlike CATMAINT, we observed higher suspend time with DSNTIJEN. In this case it might be effective to add additional pages to the buffer pools, when real storage is available. In this section we discuss how assigning larger buffer pools can effect the DSNTIJEN job.

The first measurement compares elapsed time and CPU time for DSNTIJEN job by changing BP0 size. The other buffer pools sizes are fixed to original setting during this measurement. See Figure 11-10.

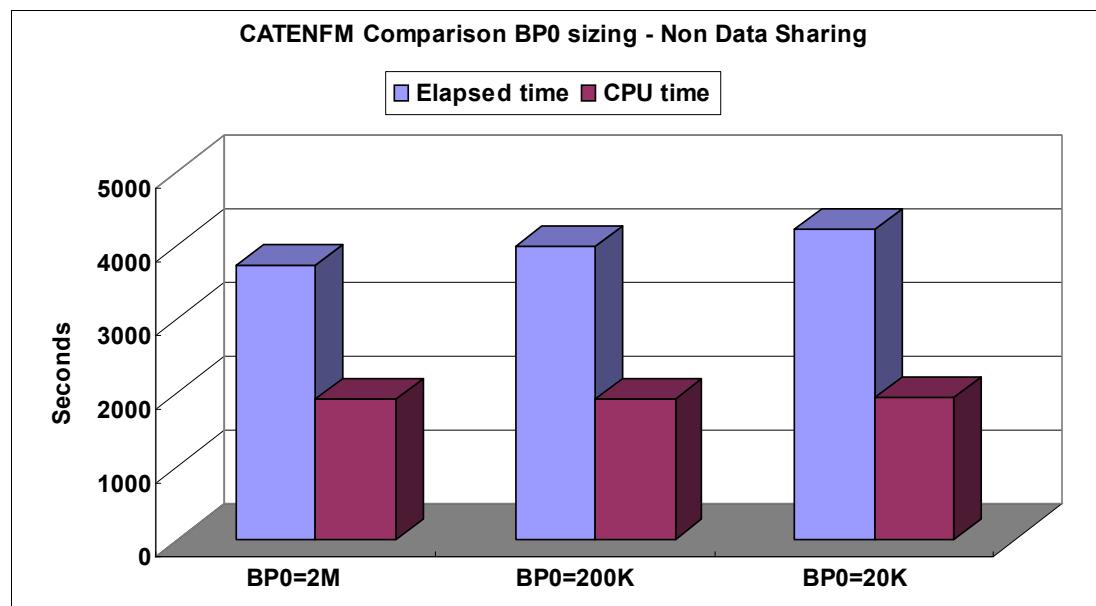


Figure 11-10 CATEENFM elapsed time and CPU time changing BP0 size

The results show that the DSNIJEN job can benefit in elapsed time from a larger BP0:

- ▶ Using BP0 with 200K pages improved elapsed time by 13.7% and CPU time by 1.5% compared with BP0 set to 20K pages.
- ▶ Using BP0 with 2M pages improved elapsed time by 19% and CPU time by 2% compared with BP0 set to 20K pages.

The reductions are mostly for processing the SPT01 page set, reducing the suspend time accounted as synchronous read I/O (see Table 11-1). Especially if you have enabled package stability in DB2 9 and have large SPT01 page set, you might consider assigning additional pages to BP0.

Table 11-1 Details of buffer pool statistics

BP0 size	20K	200K	2M
Getpages	111,934,000	111,937,000	111,935,000
Buffer updates	43,873,707	43,876,688	43,875,211
Synchronous write	10	10	10
Synchronous read	1,674,933	53,564	51,186
Asynchronous read	95	95	95
CPU time	564	513	517
Elapsed time	1,915	1,717	1,570

Figure 11-11 gives measurement results changing BP8K0 size. The BP0 size was fixed to 2M where we had the best performance with BP0 measurement. The buffer pools other than 4 KB and 8 KB are fixed to the original setting for the measurement.

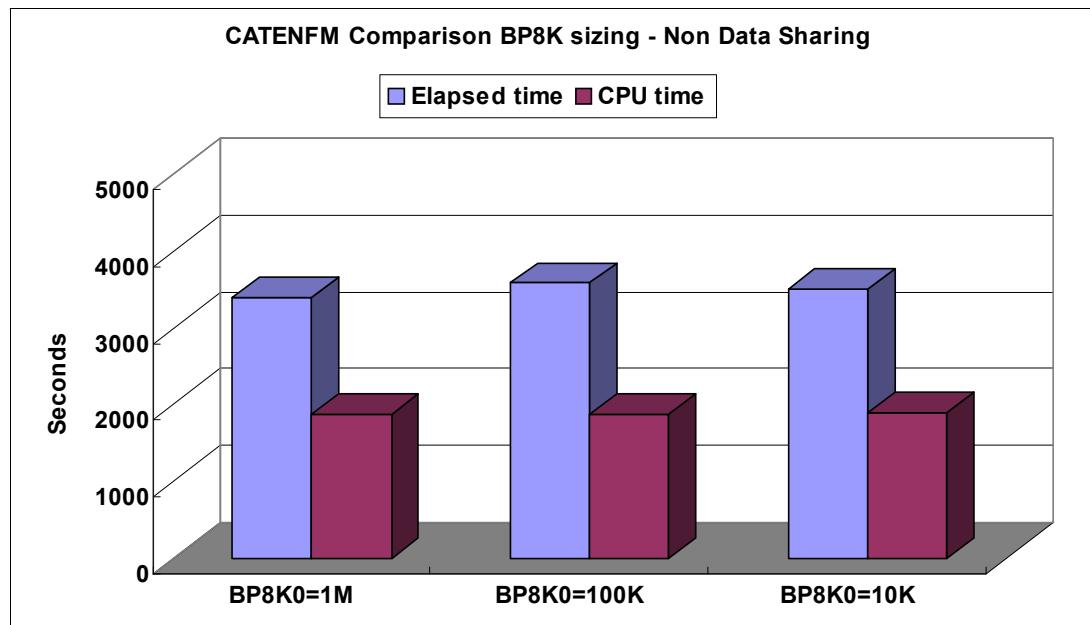


Figure 11-11 CATEENFM elapsed and CPU time changing BP8K0 size

The measurement results for BP8K0 sizes show small or no benefit when assigning larger buffer pool size:

- ▶ Increasing BP8K0 to 100K pages, we observe 4.5% improvement in elapsed time and 0.7% improvement in CPU time compared with BP8K0 set to 10K pages.
- ▶ Increasing BP8K0 to 1M pages, we observe 2.1% improvement in elapsed time and 0.7% degradation in CPU time compared with BP8K0 set to 10K pages.

Although BP8K0 is assigned to many DB2 catalog tables in SYSDBASE, as it is shown in Figure 11-9 on page 315, the process related to BP8K0 is more CPU intensive than I/O intensive. This results in no significant difference to the overall results.

Tip: If you have additional real storage to assign additional pages to buffer pool of the DB2 catalogs and directory during your migration, consider assigning them against BP0. That is where it is expected to get the most benefits based on our measurement results.

Skip level migration

Skip level migration allows you to jump straight from DB2 V8 to DB2 10, without going through DB2 9. In Figure 11-12, we show the differences between a skip level migration and a normal 2 step migrating to DB2 10 by DB2 9.

The measurements includes CATMAINT job migrating from DB2 V8 to DB2 10 CM8 and DSNTIJEN job migrating from DB2 10 CM8 to DB2 10 ENFM. The measurements were done using a non-data sharing environment and BP0 was set to 2M pages. The other buffer pools are set to our initial settings.

See Figure 11-12 and Figure 11-13 respectively for elapsed time and CPU time comparison.

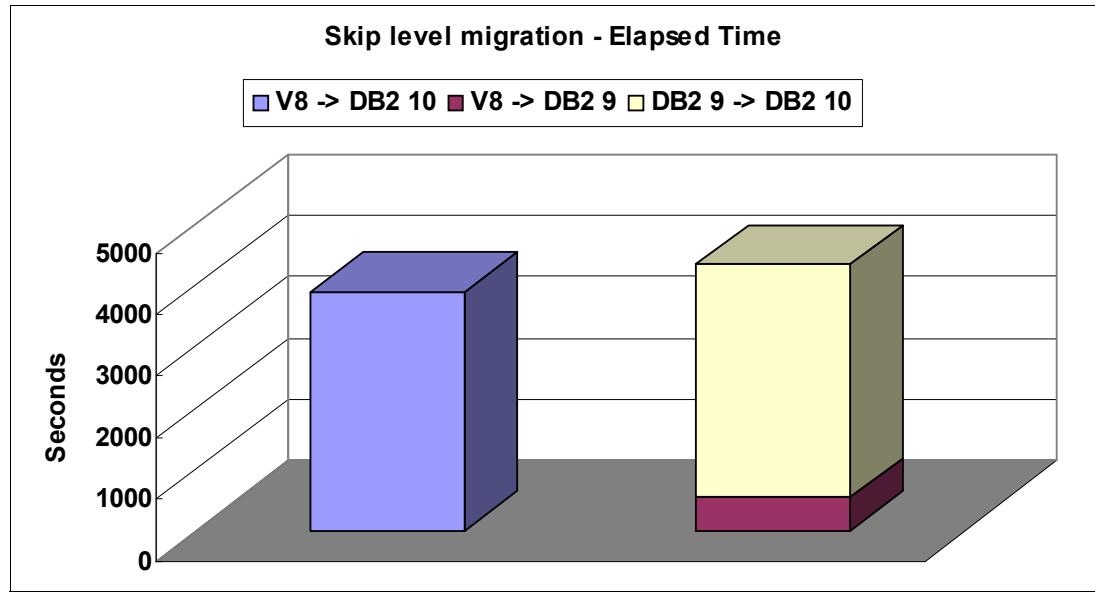


Figure 11-12 Skip level migration - Elapsed time

The results show 11% improvement in elapsed time for skip level migration, compared to migration from DB2 V8 to DB2 10 catalog by DB2 9 catalog. Skip level migration process and DB2 9 to DB2 10 catalog migration process show very similar elapsed time.

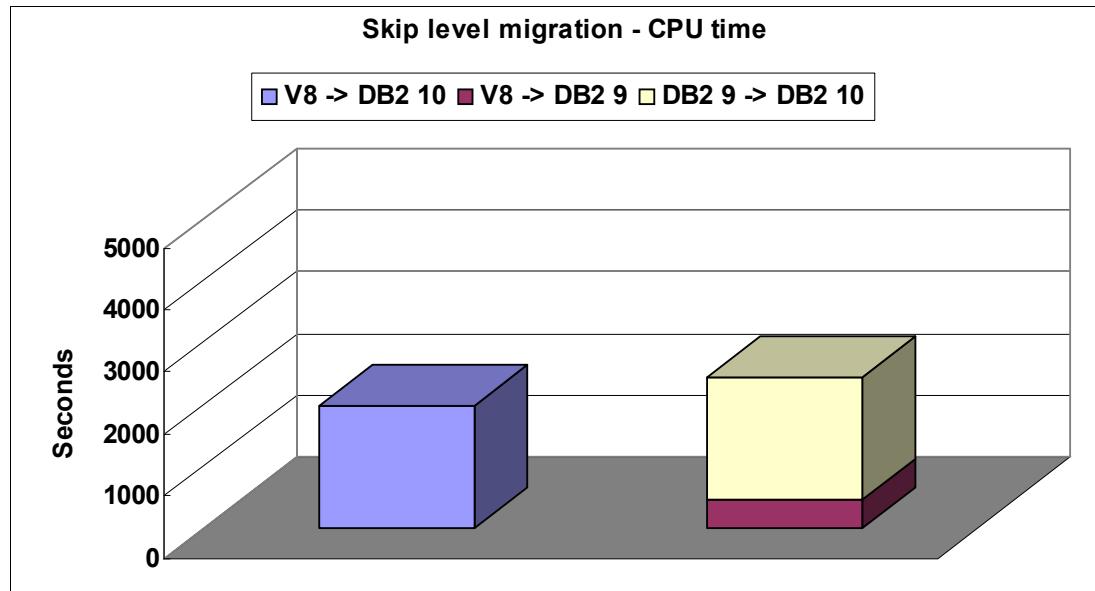


Figure 11-13 Skip level migration - CPU time

The CPU time results were more significant. The measurements show 18% improvement with skip level migration, compared with migration from DB2 V8 to DB2 10 catalog by DB2 9 catalog. When we compare the skip level migration with DB2 9 to DB2 10 catalog migration process, the results are comparable, with a difference of less than 1% in CPU time.

Considering all the changes done to the catalog in DB2 9 and DB2 10 as well as DB2 V8 to DB2 9, the measurement results for skip level migration show good results. We observed comparable elapsed and CPU time to the DB2 9 migration.

Data sharing measurements

The data sharing measurements were done using a 2-way data sharing environment. The intent is to verify if the environment difference influences the results for the overall execution of the 2 steps: CATMAINT and CATENFM. There is no GBP dependency during the measurement, and no other application was running on the system during the measurements. BP0 was set to 2 million pages and other buffer pools set to the initial settings.

Figure 11-14 shows the total elapsed time results from the data sharing measurements: no significant difference is found by comparing with the non-data sharing results.

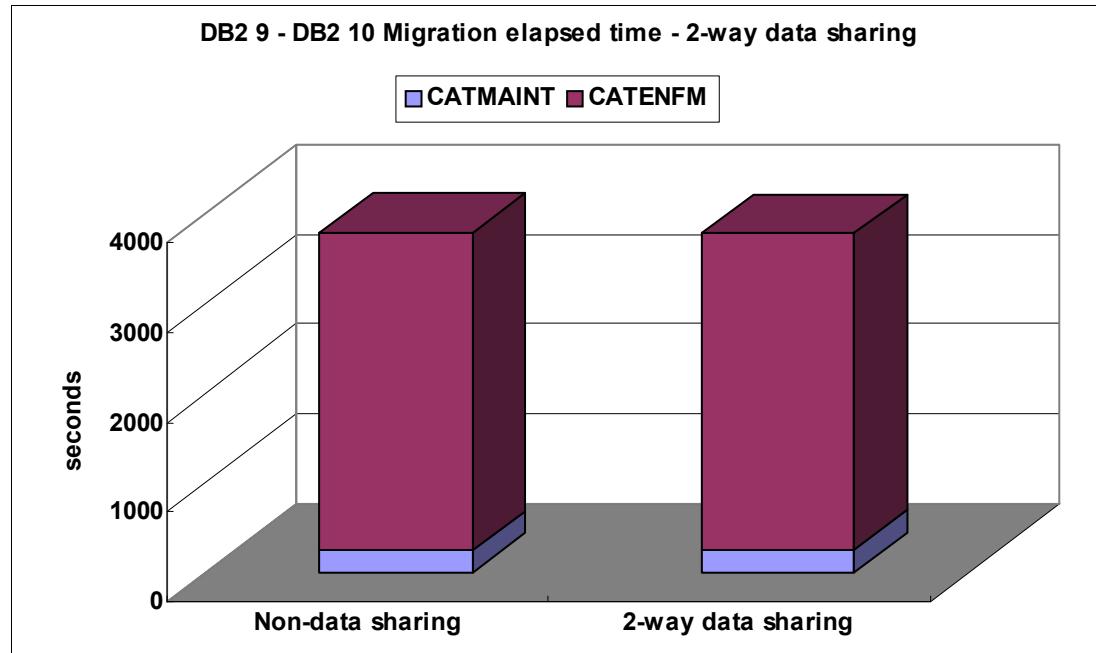


Figure 11-14 Migration elapsed time - 2-way data sharing

Figure 11-15 shows the CPU time comparison from the same measurements. No significant difference is found in CPU time by comparing with the non-data sharing results.

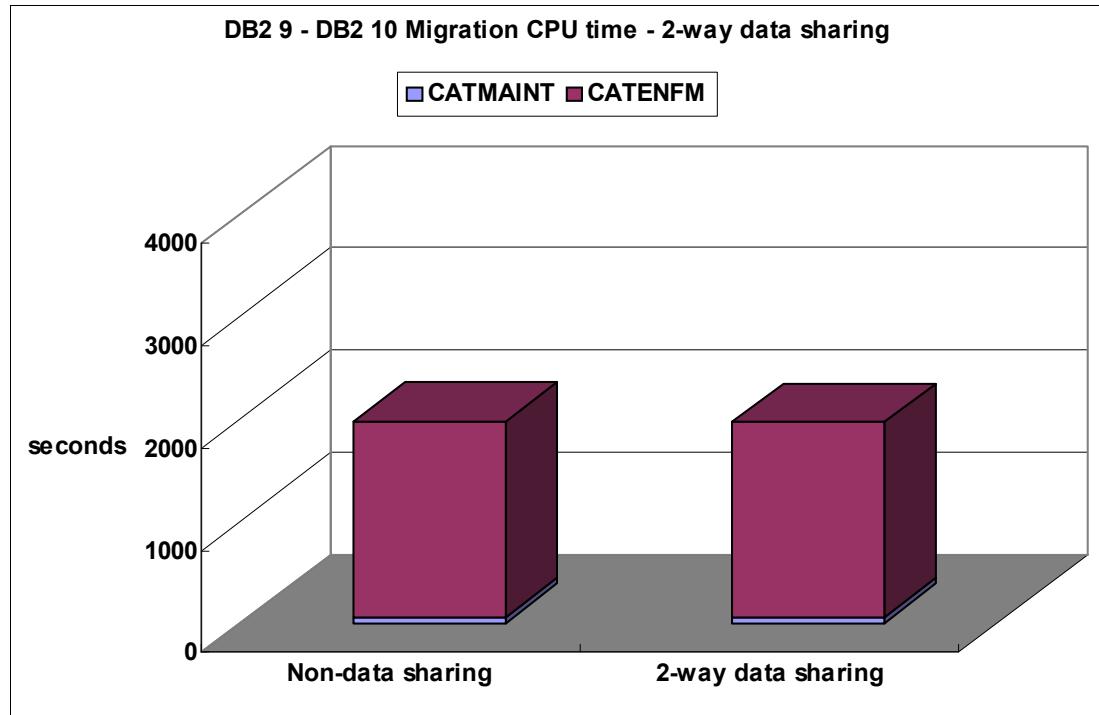


Figure 11-15 Migration CPU time - 2-way data sharing

You might expect some overhead when the catalog tables are GBP-dependent. But because the online REORG process is done against shadow data sets, you can expect not to get significant degradation. Consider running the DSNTIJEN job when there is small or no other activity in the group.

11.3.6 Conclusions and considerations

Expect a longer execution time for the DB2 10 migration process because of the catalog restructure changes during the ENFM job. If you are using access plan stability or you already have a large DB2 directory SPT01 page set, you can expect a longer time for the DSNTIJEN job execution. Keep in mind that it might take additional time if you are enabling the DB2 directory SPT01 compression functionality in DB2 9.

If your migration process takes more than your processing time window, Consider halting your DSNTIJEN job using DSNTIJNH, which will run a DSNUTILB with the CATENFM HALTENFM option. The DSNTIJNH job will halt your DSNTIJEN at the end of whichever step the job is running. You do not need to prepare another job to restart the DSNTIJEN processing, you can just re-run the DSNTIJEN job, which will restart the process from wherever you left it off.

Our measurement results showed better performance with BP0 set to 2M pages for the C1 company catalog scenario. Appropriate size can vary depending on your catalog size. Additional pages to BP0 are generally expected to help reduce DSNTIJEN job execution time.

Consider doing as much cleanup work as possible on your catalog and directory prior to starting the migration process. This cleanup work helps to identify problems before the migration and reduces the amount of data that needs to be processed.

Consider using the queries provided in the SDSNSAMP(DSNTESQ) member to check the consistency of the DB2 catalogs.

The premigration job DSNTIJPA must be run on your DB2 9 or DB2 V8 system to do an overall health check prior to migrating to DB2 10. DSNTIJPA was delivered to DB2 9 or DB2 V8 customers by the service stream in APAR PM04968. The equivalent job that shipped with DB2 10 is DSNTIJPM.



Monitoring and Extended Insight

DB2 performance traces and functions and DB2 performance tools for z/OS help with analyzing performance data, provide guidance to optimize queries, and maintain high availability by sensing and responding to situations that might result in database failures and system outages. In this chapter we concentrate on the changes that DB2 10 and OMEGAMON PE bring in this area.

In this chapter, we discuss the following topics:

- ▶ DB2 10 enhanced instrumentation
- ▶ Enhanced monitoring support
- ▶ OMEGAMON PE Extended Insight

For information about other IBM DB2 for z/OS performance management tools, see this website:

<http://www.ibm.com/software/data/db2imstools/solutions/performance-mgmt.html>

12.1 DB2 10 enhanced instrumentation

DB2 instrumentation includes the following enhancements in DB2 10:

- ▶ One minute statistics trace interval:
Statistics trace records are generated to SMF every minute, regardless of what you specify in DSNZPARM.
- ▶ IFCID 359 for index leaf page split:
A new IFCID helps you to monitor index leaf page splits.
- ▶ Separate DB2 latch and transaction lock waits in Accounting class 8:
DB2 10 externalizes both types of waits in two different fields.
- ▶ Storage statistics for DIST address space:
You can now monitor storage used by the DIST address space.
- ▶ Accounting: zAAP and zIIP SECP values:
zAAP on zIIP, zAAP, and zIIP SECP values are available.
- ▶ Package accounting information with rollup:
Package accounting statistics are also rolled up.
- ▶ DRDA remote location statistics detail:
There is more granularity in monitoring DDF locations.

Further information is provided in this section.

12.1.1 One minute statistics trace interval

As processors are getting faster and faster, more things can happen in a shorter time. For example, on a uniprocessor z10, we have the equivalent of ~60 billion instructions occurring in a single minute. That is a lot of instructions that can happen in a minute. The current default of 5 minutes is too long to see trends occurring in the system.

To help to capture events that happen inside DB2 for performance or problem diagnosis, DB2 10 always generates an SMF type 101 trace record (statistics trace) every one minute interval, no matter what you specify in the STATIME DSNZPARM parameter. This setting applies to selective statistics records critical for performance problem diagnosis.

This trace interval will not severely impact SMF data volumes, because only a few thousand records are produced per DB2 subsystem per day.

12.1.2 IFCID 359 for index leaf page split

Index leaf page splits can cause performance problems, especially in a data sharing environment. So, DB2 10 introduces a new trace record, IFCID 359, to help you to monitor leaf page splits. IFCID 359 records when index page splits occur and on what indexes.

12.1.3 Separate DB2 latch and transaction lock waits in Accounting class 8

Prior to DB2 10, plan and package accounting IFCID data does not differentiate between the time threads wait for locks and the time threads wait for latches. Accounting records produce a single field with the total for both types of wait time reported.

DB2 10 externalizes both types of waits in two different fields in all relevant IFCID records, IFCID 3, IFCID 239, and IFCID 316.

OMEGAMON PE V5.1 externalizes both counters in both the Accounting Detail report and Accounting Trace, as shown in Figure 12-1.

Report:				Trace:			
PACKAGE	AVERAGE TIME	AVG.EV	TIME/EVENT	DSNTEP2	TIME	EVENTS	TIME/EVENT
LOCK/LATCH	5.954479	23.6K	N/C	LOCK/LATCH	0.000000	0	N/C
IRLM LOCK+LATCH	5.000000	23.0K		IRLM LOCK+LATCH	0.000000	0	N/C
DB2 LATCH	0.954479	0.6K		DB2 LATCH	0.000000	0	N/C
SYNCHRONOUS I/O	1:03:11.9993	557.7K	0.033630	SYNCHRONOUS I/O	0.000000	0	N/C
OTHER READ I/O	58:53.212253	260.8K	N/C	OTHER READ I/O	0.000000	0	N/C
OTHER WRITE I/O	0.000000	0.00	N/C	OTHER WRITE I/O	0.000000	0	N/C
SERV.TASK SWITCH	1.148303	6.00	0.887426	SERV.TASK SWITCH	0.000191	2	0.000095
ARCH.LOG(QUIESCE)	0.000000	0.00	N/C	ARCH.LOG(QUIESCE)	0.000000	0	N/C
ARCHIVE LOG READ	0.000000	0.00	N/C	ARCHIVE LOG READ	0.000000	0	N/C
DRAIN LOCK	0.000000	0.00	N/C	DRAIN LOCK	0.000000	0	N/C
CLAIM RELEASE	0.000000	0.00	N/C	CLAIM RELEASE	0.000000	0	N/C
PAGE LATCH	0.000000	0.00	N/C	PAGE LATCH	0.000000	0	N/C
NOTIFY MESSAGES	0.000000	0.00	N/C	NOTIFY MESSAGES	0.000000	0	N/C
GLOBAL CONTENTION	0.000000	0.00	N/C	GLOBAL CONTENTION	0.000000	0	N/C
TCP/IP LOB	0.000000	0.00	N/C	TCP/IP LOB	0.000000	0	N/C
TOTAL CL8 SUSPENS.	2:02:12.2243	842.1K	0.318229	TOTAL CL8 SUSPENS.	0.000191	2	0.000095

Figure 12-1 Accounting suspend times

Prior to DB2 10, field IRLM LOCK+LATCH shows the accumulated elapsed time spent by the package or DBRM waiting for lock and latch suspensions. DB2 10 breaks this counter down. Field IRLM LOCK+LATCH shows the accumulated elapsed time waiting in IRLM for locks and latches. Field DB2 LATCH now records the accumulated elapsed time waiting for latches in DB2.

12.1.4 Storage statistics for DIST address space

DB2 Version 7 introduced two IFCIDs to help manage and monitor the virtual storage usage in the DBM1 address space. IFCID 225 provides summary storage information, and IFCID 217 provides more detailed information. Both IFCIDs have become key components for performance analysis and system tuning.

DB2 10 restructures IFCID 225 to take into account the DB2 10 virtual and real storage below and above the 2 GB bar mapping. The trace record is also now divided into data sections for a more logical grouping of data. IFCID 225 also contains information about storage in the DIST address space.

IFCID 217 is also overhauled. Duplicate data with IFCID 225 is now removed. Enable both IFCID 225 and IFCID 217 to generate a detail system storage profile. However, in most cases IFCID 225 is sufficient for general monitoring and reporting.

Example 12-1 shows the syntax that you can use to report these IFCIDs using OMEGAMON PE.

Example 12-1 OMEGAMON PE RECTRACE report syntax

```
/*-
//PE      EXEC PGM=FPECMAIN
//STEPLIB DD DISP=SHR,DSN=OMEGASYS.DBOA.BASE.RKANMOD
//INPUTDD DD DISP=SHR,DSN=SMFDATA.DB2RECS.G5383V00
//JOBSUMDD DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//ACRPTDD  DD SYSOUT=*
//UTTRCDD1 DD SYSOUT=*
//SYSIN    DD * *
```

```
GLOBAL  
    TIMEZONE (+ 05:00)  
RECTRACE  
TRACE  
    LEVEL(SHORT)  
INCLUDE(SUBSYSTEM(DB0A))  
INCLUDE(PRIMAUTH(DB2R1))  
INCLUDE (IFCID(225))  
EXEC  
/*
```

The report created by this example is represented in Example 12-2. This example illustrates the information collected by IFCID 225.

Example 12-2 OMEGAMON PE record trace report example

1 LOCATION: DBOA OMEGAMON XE FOR DB2 PERFORMANCE EXPERT (V5R1) PAGE: 1-9
 GROUP: N/P REQUESTED FROM: NOT SPECIFIED
 MEMBER: N/P TO: NOT SPECIFIED
 SUBSYSTEM: DBOA ACTUAL FROM: 03/10/11 13:02:00.000
 DB2 VERSION: V10 PAGE DATE: 03/10/11

OPRIMAUTHTYPE	INSTANCE	END_USER	WS_NAME	TRANSACT	
ORIGAUTHTYPE	CONNTYPE	RECORD TIME	DESTNO ACE IFC	DESCRIPTION ID	DATA
PLANNAME	CORRNMBR	TCB CPU TIME			
N/P	N/P	C772654CC707 N/P	N/P		N/P
N/P	N/P	'BLANK'	13:02:00.00566165	25794 1 225	STORAGE MGR NETWORKID: DBOA
N/P	N/P		N/P		LUNAME: DBOA LUWSEQ: 1
ADDRESS SPACE SUMMARY - DBM1					
EXTENDED REGION SIZE (MAX) : 1521483776 24-BIT LOW PRIVATE : 221184 24-BIT HIGH PRIVATE : 462848 31-BIT EXTENDED LOW PRIVATE : 69500928 31-BIT EXTENDED HIGH PRIVATE : 43126784 CURRENT HIGH ADDRESS 24-BIT PRIVATE REGION : X'0003C000' CURRENT HIGH ADDRESS 31-BIT PRIVATE REGION : X'2AFE9000' 31-BIT RESERVED FOR MUST COMPLETE : 152148377 31-BIT RESERVED FOR MVS : 25890160 STORAGE CUSHION WARNING TO CONTRACT : 152148377 TOTAL 31-BIT GETMAINED STACK : 5308416 TOTAL 31-BIT STACK IN USE : 5226496 TOTAL 31-BIT VARIABLE POOL : 14163968 TOTAL 31-BIT FIXED POOL : 86016 TOTAL 31-BIT GETMAINED : 3570740 AMOUNT OF AVAILABLE 31-BIT : 1408851968					
TOTAL 64-BIT VARIABLE POOL : 16097280 TOTAL 64-BIT FIXED : 7532544 TOTAL 64-BIT GETMAINED : 254721328 TOTAL 64-BIT PRIVATE FOR STOR MANAG: 1400832					
REAL 4K FRAMES IN USE : 55957 AUXILIARY SLOTS IN USE : 0 64-BIT REAL 4K FRAMES IN USE : 34962 64-BIT 4K AUX SLOTS IN USE : 0 HWM 64-BIT REAL 4K FRAMES IN USE : 34962 HWM 64-BIT AUX SLOTS IN USE : 0					
ADDRESS SPACE SUMMARY - DIST					
EXTENDED REGION SIZE (MAX) : 1521483776 24-BIT LOW PRIVATE : 245760 24-BIT HIGH PRIVATE : 262144 31-BIT EXTENDED LOW PRIVATE : 13570048 31-BIT EXTENDED HIGH PRIVATE : 17080320 CURRENT HIGH ADDRESS 24-BIT PRIVATE REGION : X'00042000' CURRENT HIGH ADDRESS 31-BIT PRIVATE REGION : X'261F1000' 31-BIT RESERVED FOR MUST COMPLETE : 152148377 31-BIT RESERVED FOR MVS : 26040960 STORAGE CUSHION WARNING TO CONTRACT : 152148377 TOTAL 31-BIT GETMAINED STACK : 4124672 TOTAL 31-BIT STACK IN USE : 4124672 TOTAL 31-BIT VARIABLE POOL : 761856 TOTAL 31-BIT FIXED POOL : 86016 TOTAL 31-BIT GETMAINED : 45210 AMOUNT OF AVAILABLE 31-BIT : 1490829312					
TOTAL 64-BIT VARIABLE POOL : 1466368 TOTAL 64-BIT FIXED : 118784 TOTAL 64-BIT GETMAINED : 0 TOTAL 64-BIT PRIVATE FOR STOR MANAG: 1400832					
REAL 4K FRAMES IN USE : 5019 AUXILIARY SLOTS IN USE : 0 64-BIT REAL 4K FRAMES IN USE : 605 64-BIT 4K AUX SLOTS IN USE : 0 HWM 64-BIT REAL 4K FRAMES IN USE : 605 HWM 64-BIT AUX SLOTS IN USE : 0					

OMEGAMON PE V5.1 externalizes the new DIST address space storage statistics in two new trace blocks in the Statistics Short and Long reports, as shown in Figure 12-2 and Figure 12-3.

DIST STORAGE ABOVE 2 GB	
GETMAINED STORAGE	(MB)
FIXED STORAGE	(MB)
VARIABLE STORAGE	(MB)
STORAGE MANAGER CONTROL BLOCKS	(MB)

Figure 12-2 Statistics, DIST storage above 2 GB

DIST AND MVS STORAGE BELOW 2 GB	
TOTAL DIST STORAGE BELOW 2 GB	(MB)
TOTAL GETMAINED STORAGE	(MB)
TOTAL VARIABLE STORAGE	(MB)
NUMBER OF ACTIVE CONNECTIONS	
NUMBER OF INACTIVE CONNECTIONS	
TOTAL FIXED STORAGE	(MB)
TOTAL GETMAINED STACK STORAGE	(MB)
TOTAL STACK STORAGE IN USE	(MB)
STORAGE CUSHION	(MB)
24 BIT LOW PRIVATE	(MB)
24 BIT HIGH PRIVATE	(MB)
24 BIT PRIVATE CURRENT HIGH ADDRESS	
31 BIT EXTENDED LOW PRIVATE	(MB)
31 BIT EXTENDED HIGH PRIVATE	(MB)
31 BIT PRIVATE CURRENT HIGH ADDRESS	
EXTENDED REGION SIZE (MAX)	(MB)

Figure 12-3 Statistics, DIST storage below 2 GB

12.1.5 Accounting: zAAP and zIIP SECP values

z/OS 1.11 is enhanced with a new function that can enable System z Application Assist Processor (zAAP) eligible workloads to run on a System z Integrated Information Processor (zIIP). This function can enable you to run zIIP- and zAAP-eligible workloads on the zIIP. This capability is ideal for customers without enough zAAP or zIIP-eligible workload to justify a specialty engine today. The combined eligible workloads can make the acquisition of a zIIP cost effective. This capability is also intended to provide more value for customers having only zIIP processors by making Java and XML-based workloads eligible to run on existing zIIPs.

This capability is available with z/OS 1.11 (and z/OS 1.9 and 1.10 with PTF for APAR OA27495) and all System z9 and System z10 servers. This capability does not provide an overflow so additional zAAP eligible workload can run on the zIIP, it enables the zAAP eligible work to run on zIIP when no zAAP is defined.

APAR PK51045 renames the redirection eligible zIIP CPU time (IIPCP CPU TIME) to SECP CPU and the consumed zIIP CPU time (IIP CPU TIME) to SE CPU TIME, to more accurately reflect what they are, Specialty Engine times which can have both zAAP and zIIP components. These fields are externalized in the OMEGAMON PE DB2 Accounting reports.

Beginning with DB2 10, the possible redirection value SECP, which used to indicate either the estimated (PROJECTCPU) or zIIP overflow to CP if zIIP is too busy, is no longer supported and is always zero. However, the actual redirected CPU time continues to be available in the SE CPU time field. The reason is that z/OS cannot provide possible redirection values for zIIP on zAAP engines to DB2, so the value cannot represent all the specialty engines as the name implies. SECP CPU is still reported for DB2 9 and earlier; however, it lists only the possible redirection for zIIP eligible processes. SE CPU TIME remains the actual CPU time consumed for both zAAP and zIIP processors.

In DB2 10, you need to review the RMF Workload Activity reports to get an indication of how much work is eligible for zIIP or zAAP processing (PROJECTCPU) or how much work overflowed to CP because zIIP or zAAP was too busy. Look at the AAPCP (for zAAP) and IIPCP (for zIIP) in the APPL% section of the RMF workload activity report Service or Reporting class section as shown in the following example:

```
---APPL %---
CP      131.62
AAPCP   0.00
IIPCP   79.00

AAP     0.00
IIP     0.00
```

12.1.6 Package accounting information with rollup

DB2 currently provides the facility to accumulate accounting trace data. A *rollup record* is written with accumulated counter data for the following type of threads:

- ▶ Query parallelism child tasks if the DSNZPARM PTASKROL parameter is set to YES
- ▶ DDF and RRSAF threads if the DSNZPARM ACCUMACC parameter is greater than or equal to 2

For query parallelism child tasks, a rollup record is written with accumulated counter data when the parent task (agent) deallocates on an originating DB2 or when an accumulating child task deallocate on an assisting DB2. The rollup data is an accumulation of all the counters for that field for each child task that completed and deallocated.

For DDF and RRSAF threads, a rollup record is written with accumulated counter data for a given user when the number of occurrences for that user reaches the DSNZPARM value for ACCUMACC. The user is the concatenation of the following values, as defined by the DSNZPARM ACCUMUID parameter:

- ▶ User user ID (QWHEUID, 16 bytes)
- ▶ User transaction name (QWHCEUTX, 32 bytes)
- ▶ User workstation name (QWHCEUWN, 18 bytes)

Accounting statistics are rolled up only at the plan level (Accounting class 1, 2, and 3) in releases prior to DB2 10.

DB2 10 enhances accounting rollup for the package level, (accounting class 7, 8 and 10 if available).

This change impacts IFCID 003, IFCID 239, IFCID 147, IFCID 148, and SMF type 100 and 101 records (subtypes in header).

12.1.7 DRDA remote location statistics detail

Prior versions of DB2 collect statistics for all locations accessed by DRDA and group them under one collection name, DRDA REMOTE LOCS, and these statistics are reported in the Statistics Detail reports in one single block, DRDA REMOTE LOCATIONS. DB2 10 introduces a number of enhancements to IFI tracing to address these challenges.

Because many thousands of remote clients can potentially be in communication with a DB2 subsystem, DB2 10 introduces a statistics class 7 trace to externalize DRDA statistics data by location. The new statistics class 7 trace triggers a new IFCID, 365, to be activated.

When a statistics trace is started with class 7 specified in the class list, the location data is only written out at the interval specified by the STATIME DSNZPARM parameter. During normal DB2 statistics trace processing (CLASS 1, 2, 4, 5, or 6), only the statistics for the location named DRDA REMOTE LOCS are externalized to the specified statistics destination, which defaults to SMF. The information is written to the specified destination every minute.

To get the new detail location statistics, you must either specify CLASS(7) or IFCID(365) on a -START TRACE or -MODIFY TRACE command, which activates a new or modifies an existing statistics trace. DB2 then writes new IFCID 365 records to the specified destination of the statistics trace for the remote locations that are communicating with the subsystem.

The new record includes only the statistics for those locations with activity since a record was generated. The statistics for up to 95 locations are written in one record with more than one record being written until all locations are retrieved. The default DRDA location, DRDA REMOTE LOCS, which has all location statistics, is still written every time the default statistics trace classes are written, but it is not written with the other detail location statistics in the IFCID 365 trace.

When a monitor trace is started for IFCID 365, multiple IFCID 365 records are returned in response to a READS request. The issuer of the READS must provide a buffer of suitable size. The number of IFCID 365 records written to the buffer depends on the size of the buffer passed as a parameter of the READS. The buffer must be large enough to hold at least one QW0365 location detail section and the QW0365HE header section. Again, only the locations that have activity since the last READS were processed are returned.

DDF continues to capture statistics about all DRDA locations in one location named DRDA REMOTE LOCS. The statistics for this default DRDA location is the only location statistics written with the default statistics trace data, which is a change from DB2 9 where the statistics for up to 32 locations were written.

DDF now captures statistics about each location that it is in communication with as a server or a requester. The statistics for these locations is written every time the STATIME interval has elapsed. The statistics detail for up to 75 locations is written in one IFCID 365 record. Multiple records are produced until the detail statistics for all locations showing activity are written.

DDF continues to capture statistics at the accounting level for every location referenced by the thread during a single accounting interval or a rollup of multiple accounting intervals. The information is written every time an accounting record is requested to be sent to disk.

IFCID 365 trace data can be accessed through the monitor interface by starting a user defined class monitor trace with IFCID 365 specified. The location statistics are returned in response to a READS request.

STATIME: The STATIME subsystem parameter, defined in the installation tracing panel, DSNTIPN, applies only to IFCIDs 0105, 0106, 0199, and 0365.

In addition, some location statistics counters collected are private protocol only and need to be removed from the statistics that are captured. DB2 10 also restructures trace records to expand DDF counters and remove redundant private protocol counters. The layout for the DRDA remote locations reporting has not changed in OMEGAMON PE statistics reports. Obsolete fields are shown as N/A.

12.2 Enhanced monitoring support

Although DB2 for z/OS has support for problem determination and performance monitoring at the statement level for both dynamic and static SQL, primarily through tracing, the following limitations apply:

- ▶ Identifying the specific SQL statements involved in a problem (for example deadlocks, timeouts, and unavailable resource conditions) can be expensive and costly. In addition, the process to narrow the application in question can be time-consuming and can involve resources from many different teams including (DBA, application developers, system administrators, and users). You might need to examine traces after the original problem occurs because insufficient information was captured at the point where the problem first occurred. Trace and log files can be huge and can change the timing behavior of applications.
- ▶ For distributed workloads, low priority or poorly behaving client applications can monopolize DB2 resources and prevent high-priority applications from executing. There is no way to prioritize traffic before it arrives into DB2. WLM classification of workload does not help because it does not occur until after the connection has been accepted and a DBAT is associated to the connection. WLM makes sure that high priority work completes more quickly, but if there is a limited number of DBATs or connections, low priority work has equal access to the threads or connections and can impact the higher priority work. (In a data sharing environment, you can use LOCATION ALIAS NAMES to connect to a subset of members but this has limited granularity.)

For distributed workloads, the number of threads, the number of connections and idle thread timeout are controlled by system level parameters (MAXDBAT, CONDBAT, and IDTHTOIN), so all distributed clients are treated equally. This situation is a problem if the client applications do not represent equal importance to the business.

DB2 10 for z/OS enhances performance monitoring support and monitoring support for problem determination for both static and dynamic SQL. This new support uses the IFI to capture and externalize monitoring information for consumption by tooling.

To support problem determination, the statement type (dynamic or static) and statement execution identifier (STMT_ID) are externalized in several existing messages (including those related to deadlock, timeout, and lock escalation). In these messages, the statement type and statement identifier are associated with thread information (THREAD-INFO) that can be used to correlate the statement execution on the server with the client application on whose behalf the server is executing the statement.

To support performance monitoring, several existing trace records that deal with statement level information are modified to capture the statement type, statement identifier, and new statement-level performance metrics. New trace records are introduced to provide access to performance monitoring statistics in real time and to allow tooling to retrieve monitoring data without requiring disk access. In addition, profile monitoring is introduced to increase granularity and to improve threshold monitoring of system level activities. The monitor profile supports monitoring of idle thread timeout, the number of threads, and the number of connections, as well as the ability to filter by role and client product-specific identifier.

12.2.1 Unique statement identifier

DB2 10 introduces a unique statement execution identifier (STMT_ID) to facilitate monitoring and tracing at the statement level. The statement ID is defined at the DB2 for z/OS server, returned to the DRDA application requester, and captured in IFCID records for both static and dynamic SQL.

For dynamic statement, the statement identifier is available when dynamic statement cache is enabled. For static statement, the statement identifier matches the STMT_ID column value in the SYSIBM.SYSPACKSTMT table. STMT_ID column is a new column in the SYSIBM.SYSPACKSTMT table.

To use the enhanced monitoring support functions, you must rebind or bind any existing pre-DB2 10 package in DB2 10 new function mode in order for the STMT_ID column to be populated and loaded into the package.

Through DRDA, the statement identifier is returned to the application requesters in addition to a 2-byte header information which includes the SQL type information (indicating dynamic or static statement), and a 10-byte compilation time in a representation of a timestamp format with the form of *yyyymmddhhmmssnnnnnn* (the last bind time for static SQL, and the time of compilation for dynamic SQL). This information is contained in DRDA MONITORID object and is returned through the DRDA MONITORRD reply. When DRDA MONITORID is returned in the DRDA MONITORRD reply data, DB2 for z/OS server also returns DRDA SRVNAM object along in the DRDA MONITORRD reply data providing the specific server.

The following IFCIDs are changed to externalize the new statement identifier:

- ▶ IFCID 58 (end SQL) is enhanced to return statement type, statement execution identifier and statement level performance metrics. For related statements such as OPEN, FETCH, and CLOSE, only the IFCID 58 of the CLOSE request provides information which reflect the statistics collected on the OPEN and FETCHes as well.
- ▶ IFCID 63 and 350 (SQL statement) are enhanced to return statement type, statement execution identifier, and the original source CCSID of the SQL statement. This new information is returned for the statement types that are candidates to be in the DB2 dynamic statement cache (SELECT, INSERT, UPDATE, DELETE, MERGE, and so on) when the dynamic statement cache is enabled.
- ▶ IFCID 124 (SQL statement record) is enhanced to return the statement type and statement execution identifier.
- ▶ IFCID 66 (close cursor) is enhanced to trace implicit close statement. A new field is added to indicate if the close statement is an explicit close statement or an implicit close statement.
- ▶ IFCID 65 (open cursor) is enhanced to trace the cursor attribute on the implicit commit request. A new field is added to indicate if implicit commit cursor attribute is specified.
- ▶ IFCID 172 and IFCID 196 (deadlock and timeout) are enhanced to return statement type and statement execution identifier for both dynamic and static statement when a deadlock or timeout condition is detected.
- ▶ IFCID 337 (lock escalation) is enhanced to return statement type and statement execution identifier for both dynamic and static statement when lock escalation condition is detected.

In addition, the THREAD-INFO token is extended to include statement type information, besides role and statement ID information. The affected messages are DSNL030I, DSNV436I, DSNL027I, DSNI031I, DSNT318I, DSNT375I, DSNT376I, DSNT377I, DSNT378I, DSNT771I, and DSNT772I.

12.2.2 New monitor class 29 for statement detail level monitoring

Monitor class 9 tracing provides real-time statement level detail per thread basis, with IFCID 124. Monitor class 29 is introduced to monitor detailed trace information in real-time for all the dynamic statements in the dynamic statement cache and all static statements currently in the EDM Pool. Monitor class 29 looks at the system-wide level rather than the thread-level.

Monitor class 29 has the following IFCIDs:

- ▶ IFCID 318 is an existing IFCID that is merely a flag to instruct DB2 to collect more detailed statistics about statements in the dynamic statement cache.
- ▶ IFCID 316 is an existing IFCID that provides detailed information about the dynamic statement cache when promoted by an IFI READS request. IFCID is also written when a statement is evicted from the dynamic statement cache.
- ▶ IFCID 400 is a new IFCID that essentially mirrors the behavior of IFCID 318.
- ▶ IFCID 401 is a new IFCID that when prompted externalizes all static statements in the EDM pool together with their detailed statistics. IFCID is also written when a statement is removed from the EDM Pool. STMTID support (as well as 401 generation) requires REBIND in NFM.

The statement identifiers and new IFCID 401 are utilized in the new *Extended Insight* feature of OMEGAMON PE V5.1 within the IBM Optim™ Performance Manager infrastructure.

Example 12-3 shows how to start the CLASS 29 monitor traces.

Example 12-3 Starting CLASS 29 traces

```
-STA TRA(MON) C(29) DEST(GTF)
```

Figure 12-4 shows the performance impact of starting the monitoring class 29 during the execution of a ITR workload. The observed degradation in throughput is close to 1%.

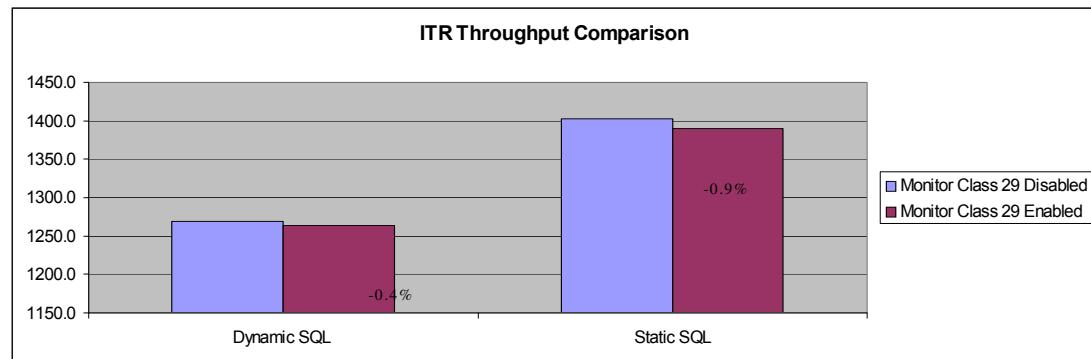


Figure 12-4 Performance impact of monitoring class 29

The same ITR workload was executed but with the performance IFCIDs started by the command in Example 12-4.

Example 12-4 Starting performance traces

```
-STA TRA(P) C(30) IFCID(58,63,65,66,350) DEST(GTF)
```

The observed results are represented in Figure 12-5. The negative impact in throughput is almost 5%; this is on the order of 5 times more than the observations for the monitor class 29.

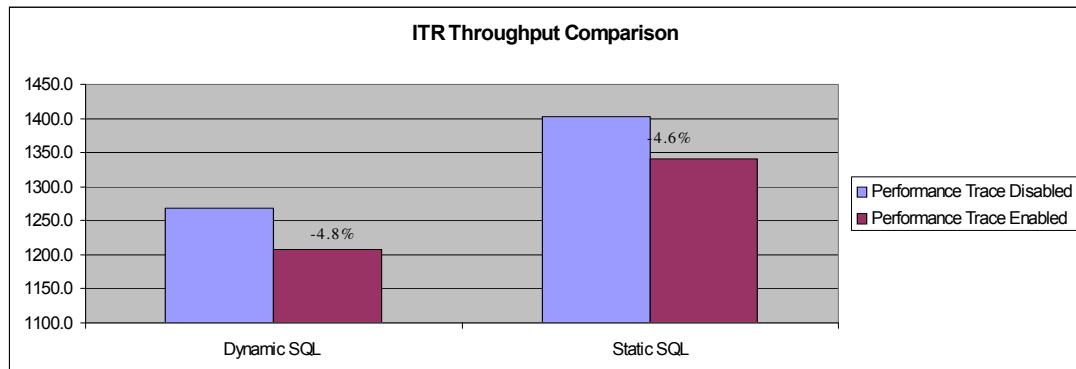


Figure 12-5 Performance impact of performance IFCIDs

12.2.3 System level monitoring

DB2 9 for z/OS introduced profiles to allow you to identify a query or set of queries. It is possible to identify SQL statements by authorization ID and IP address, or to specify combinations of plan name, collection ID, and package name. These profiles are specified in the table SYSIBM.DSN_PROFILE_TABLE. Profile tables allow you to record information about how DB2 executes or monitors a group of statements. SQL statements identified by a profile are executed based on keywords and attributes in the table SYSIBM.DSN_PROFILE_ATTRIBUTES. System parameters such as NPGTHRSH, STARJOIN and SJTABLES can be specified as keywords, providing greater granularity than DSNZPARM parameters.

In DB2 9 for z/OS, it is possible to identify SQL statements by authorization ID and IP address, or to specify combinations of plan name, collection ID, and package name. All statements so identified can be monitored. The following tables are required to monitor statements:

- ▶ SYSIBM.DSN_PROFILE_TABLE
- ▶ SYSIBM.DSN_PROFILE_HISTORY
- ▶ SYSIBM.DSN_PROFILE_ATTRIBUTES
- ▶ SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY
- ▶ SYSIBM.DSN_STATEMENT_RUNTIME_INFO

The following tables might also be required, depending on the information that DB2 is to record:

- ▶ SYSIBM.DSN_OBJECT_RUNTIME_INFO
- ▶ SYSIBM.PLAN_TABLE
- ▶ SYSIBM.DSN_STATEMENT_TABLE
- ▶ SYSIBM.DSN_FUNCTION_TABLE
- ▶ Other EXPLAIN tables that are used by optimization tools

Statement execution is controlled by the following keywords:

- ▶ NPAGES THRESHOLD
- ▶ STAR JOIN
- ▶ MIN STAR JOIN TABLES

Other keywords control the amount of monitoring data recorded.

The information collected in these tables was used by tools such as the Optimization Service Center and Optimization Expert. See *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421. These tools are now replaced by OPTIM Query Tuner and OPTIM Workload Tuner.

DB2 10 enhances support for filtering and threshold monitoring for system related activities, such as the number of threads, the number of connections, and the period of time that a thread can stay idle.

Two new scope filters on role (available through trusted context support) and client product-specific identifier are added to provide more flexibility to monitor the activities across the DB2 system. Allowing filtering by role and client product-specific identifier gives a finer degree of control over the monitor profiles.

Similarly, the addition of new function keywords related to the number of threads, the number of connections and idle thread timeout values allows thresholds (limits) that were previously available only at the system level through DSNZPARM to be enforced at a more granular level.

Together, these enhancements provide a greater flexibility and control with regard to allocating resources to particular clients, applications, or users according to their priorities or needs.

The enhancements in DB2 10 apply only to the system level monitoring related to threads and connections, not to the statement level monitoring and tuning.

Briefly, to use system level monitoring, you must first create the following tables, if they do not already exist, by running either installation job DSNTIJSG or DSNTIJOS:

- ▶ **SYSIBM.DSN_PROFILE_TABLE**

This table includes one row per monitoring or execution profile. A row can apply to either statement monitoring or system level monitoring but not both. Multiple profile rows can apply to an individual execution or process, in which case the more restrictive profile is applied.

- ▶ **SYSIBM.DSN_PROFILE_HISTORY**

This table tracks the state of rows in the profile table or why a row was rejected.

- ▶ **SYSIBM.DSN_PROFILE_ATTRIBUTES**

This table relates profile table rows to keywords, which specify monitoring or execution attributes that define how to direct or define the monitoring or execution.

- ▶ **SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY**

This table tracks the state of rows in the attributes table or why a row was rejected.

To monitor system level activities, you must take the following actions:

1. Create a profile to specify what activities to monitor.
2. Define the type of monitoring. (There are three keywords and two attribute columns.)
3. Load or reload profile tables and start the profile.
4. Stop monitoring.

To create a profile, add a row to the table SYSIBM.DSN_PROFILE_TABLE. DB2 10 provides the ROLE and PRDID columns for additional filtering. ROLE filters by the role that is assigned to the user who is associated with the thread. PRDID filters by the client product-specific identifier that is currently associated with the thread, for example JCC03570. In addition, different profiles can apply to different members of a data sharing group.

The following filtering categories are supported for system level monitoring:

- ▶ IP address (IPADDR)
- ▶ Product identifier (PRRID)
- ▶ Role and authorization identifier (ROLE, AUTHID)
- ▶ Collection ID and Package name (COLLID, PKGNAME)

The filtering criteria in different filtering categories cannot be specified together. For example, IP Address and Product ID cannot be specified together as a valid filtering criteria because IP Address and Product ID are in different filtering categories.

The newly introduced ROLE and product ID filtering scope can also only apply to the system related monitoring functions or keywords regarding threads and connections.

The product-specific identifier of the remote requester (PRRID), is in the form *pppvvrrm*, an 8-byte field with alphanumeric characters, which have the following meanings:

<i>ppp</i>	Identifies the specific database product
<i>vv</i>	Identifies the product version
<i>rr</i>	Identifies the product release level
<i>m</i>	Identifies the product modification level

When there is more than one filtering scope in the same filtering category, the filtering scope can exist without the other. For example, for the role and authorization ID filtering category, role can be filtered by itself or authorization ID can be filtered by itself, or role and authorization ID can be filtered together. The same applies to collection ID and package name filtering category. That is, for collection ID and package name filtering category, collection ID can be filtered by itself or package name can be filtered by itself, or collection ID and package name can be filtered together. These new filtering category rules only apply to the system related monitoring functions or keywords in regard to threads and connections.

Figure 12-6 shows that profiles 11 and 15 are invalid because there are different filtering categories defined in the same row. There are corresponding rows in SYSIBM.DSN_PROFILE_HISTORY that show the reason for the rejection in the STATUS column.

ROLE	AUTHID	IPADDR	PRRID	COLLID	PKGNAME	PROFILEID	PROFILE_ENABLED
DB2DEV	Tom	Null	Null	Null	Null	10	Y
DB2DEV	Null	Null	SQL09073	Null	Null	11	Y
Null	Tom	Null	Null	Null	Null	12	Y
Null	Null	Null	JCC03570	Null	Null	13	Y
Null	Null	129.42.16.152	Null	Null	Null	14	Y
Null	Null	129.42.16.152	Null	COLL1	Null	15	Y

Figure 12-6 System level monitoring - Invalid profile

Multiple qualified filtering profiles from different filtering categories can all apply if the filtering criteria matches the thread or connection for system level monitoring. For example, in Figure 12-6, profile 10 and 14 might both apply to Tom's thread or connection.

Within the same filtering category, there is only one qualified profile applied. When there is more than one profile qualified within the same filtering category, the most specific profile is chosen. In Role and Authorization ID filtering category, Role has higher precedence over Authorization. In the Collection ID and Package name filtering category, Collection ID has higher precedence over Package name. In the previous example, if profile 10 and 12 both apply to Tom's thread or connection for the same keyword, row 10 will apply, as ROLE takes precedence.

You define the type of monitoring that you want to perform by inserting a row into the table SYSIBM.DSN_PROFILE_ATTRIBUTES with the following attributes:

- ▶ Profile ID column:

Specify the profile that defines the system activities that you want to monitor. Use a value from the PROFILEID column in SYSIBM.DSN_PROFILE_TABLE.

- ▶ KEYWORDS column:

Specify one of the following monitoring keywords:

MONITOR THREADS	The number of active threads for this profile. This value cannot exceed MAXDBAT.
-----------------	--

MONITOR CONNECTIONS	The total number of connections, active plus inactive, for this profile. This value cannot exceed CONDBAT.
---------------------	--

MONITOR IDLE THREADS	The number of seconds before idle threads timeout, for this profile. This value can exceed IDTHTOIN.
----------------------	--

- ▶ ATTRIBUTE1, ATTRIBUTE2, and ATTRIBUTE3 columns:

Specify the appropriate attribute values depending on the keyword that you specify in the KEYWORDS column.

There are two levels of messaging mechanisms introduced (DIAGLEVEL1 and DIAGLEVEL2) as described later for system profile monitoring related to threads and connections. You can choose which messaging level you prefer by defining a value in the ATTRIBUTE1 column in the DSN_PROFILE_ATTRIBUTES table.

You specify the threshold for the corresponding keyword in ATTRIBUTE2.

Currently ATTRIBUTE3 does not apply to any of the system related monitoring functions, so there is no need to enter any value for the ATTRIBUTE3 column. If there is value specified in ATTRIBUTE3, the row for system level monitoring keyword, a row is inserted into SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY table to indicate that this row is invalidated.

You do no need to specify values for any other columns for SYSIBM.DSN_PROFILE_ATTRIBUTES.

MONITOR THREADS is used to monitor the total number of concurrent active threads on the DB2 subsystem. This monitoring function is subject to the filtering on IPADDR, PRDID, ROLD/AUTHID and COLLID/PKGNAME defined in SYSIBM.DSN_PROFILE_TABLE.

MONITOR CONNECTIONS is used to monitor the total number of remote connections from the remote requesters using TCP/IP, which includes the current active connections and the inactive connections. This monitoring function is subject to the filtering on the IPADDR column only in the SYSIBM.DSN_PROFILE_TABLE for remote connections. Active connections are those currently associated with an active DBAT or have been queued and are waiting to be serviced. Inactive connections are those currently not waiting and not associated with a DBAT.

MONITOR IDLE THREADS is used to monitor the approximate time (in seconds) that an active server thread must be allowed to remain idle.

It is important to note that these system level function keywords are to monitor system related activities such as the number of threads, the number of connections, and idle time of the threads, not to monitor any SQL statements activities.

ATTRIBUTE1 specifies either **WARNING** or **EXCEPTION** and the messaging level, as follows:

- **WARNING** (defaults to **DIAGLEVEL1**)
- **WARNING DIAGLEVEL1**
- **WARNING DIAGLEVEL2**
- **EXCEPTION** (defaults to **DIAGLEVEL1**)
- **EXCEPTION DIAGLEVEL1**
- **EXCEPTION DIAGLEVEL2**

ATTRIBUTE2 specifies the actual threshold value, for example, the number of active threads or total connections, or seconds for idle threads.

ATTRIBUTE3 must remain blank.

If a **WARNING** is triggered, then either message DSNT771I or DSNT772I is issued depending on the **DIAGLEVEL** specified and processing continues. However, if an **EXCEPTION** is triggered, then processing varies depending on the exception triggered.

► **MONITOR THREADS:**

Threads are either queued or suspended as follows:

- If **IPADDR** filter, then return **RC00E30506** and queue the threads.
- If **PROID**, **ROLE**, or **AUTHID** filter, then return **RC00E30507** and queue and suspend for threshold, then fail additional connection requests with **SQLCODE -30041**.
- If **COLLID** or **PKGNAME** filter, then return **RC00E30508** and queue and suspend for threshold, then fail SQL statement and return **SQLCODE -30041**.

► **MONITOR CONNECTIONS (IPADDR filtering only):**

- Issue **RC00E30504** and reject new connection requests.

► **MONITOR IDLE TREADS – thread terminated:**

- Issue **RC00E30502**.

Note that the system-wide installation parameters related to maximum connections (for example, **CONDBAT**), maximum threads (for example, **MAXDBAT**) and idle thread timeout (**IDTHTOIN**) still apply. In the event that the system-wide threshold is set lower than a monitor profile threshold, the system-wide threshold is enforced before the monitor profile threshold can apply.

When **DIAGLEVEL1** is chosen, a DSNT771I console message is issued at most once every 5 minutes for any profile threshold that is exceeded. This level of messaging provides minimum console messages activity and limited information in the message text itself. Refer to the statistics trace records to determine the accumulated occurrences of a specific profile threshold that is exceeded under a specific profile ID:

```
DSNT771I csect-name A MONITOR PROFILE condition-type CONDITION OCCURRED  
numberof-time TIME(S).
```

When **DIAGLEVEL2** is chosen, a DSNT772I console message is issued at most once every 5 minutes for each unique occurrence of the profile threshold in a specific Profile ID that is exceeded. This level of messaging provides more information in the message text itself which includes the specific profile ID and the specific reason code. You can also refer to the

statistics trace records to determine the accumulated occurrences of a specific profile threshold that is exceeded under a specific profile ID.

```
DSNT772I csect-name A MONITOR PROFILE condition-type CONDITION OCCURRED
numberof-time TIME(S) IN PROFILE ID=profile-id WITH PROFILE FILTERING
SCOPE=filtering-scope WITH REASON=reason
```

For both message levels, when a profile warning or exception condition occurs, a DB2 statistics class 4 IFCID 402 trace record is written at a statistical interval which is defined in the STATIME DSNZPARM. Each statistics trace record written can contain up to 500 unique profiles. Multiple trace records can be written if there are more than 500 unique profiles whose profile thresholds are exceeded in a given STATIME interval.

Figure 12-7 shows sample rows from SYSIBM.DSN_PROFILE_ATTRIBUTES. The REMARKS column is not shown.

ProfileID	Keywords	Attribute1	Attribute2	Attribute3	Attribute Timestamp
1	MONITOR THREADS	Exception_dia glevel2	10 Queue 11-20 *Reject 21st		2009-12-19...
2	MONITOR CONNECTIONS	Warning	50		2009-12-19...
3	MONITOR IDLE THREADS	Exception_dia glevel1	300		2009-12-21...

Figure 12-7 System level monitoring - Attributes table

The first row indicates that DB2 monitors the number of threads that satisfy the scope that is defined by PROFILEID 1 in SYSIBM.DSN_PROFILE_TABLE. When the number of the threads in the DB2 system exceeds 10 that is defined in ATTRIBUTE2 column, a DSNT772I message is generated to the system console (if there is no other DSNT772I message being issued within last 5 minutes due to this particular profile threshold in PROFILEID 1 is exceeded) and DB2 queues or suspends the number of any new connection request up to 10, the defined exception threshold in ATTRIBUTE2, as EXCEPTION_DIAGLEVEL2 is defined in the ATTRIBUTE1 column.

When the total number of threads that are queued or suspended reaches 10, DB2 begins to fail the connection request with SQLCODE -30041 (up to 10 threads for this profile, PLUS up to 10 queued connections for this profile). The 21st connection for this profile receives a -30041, unless the profile filters on IPADDR. In such a case, any number of connections are allowed, up to MONITOR CONNECTIONS, if such a row is provided for this profile with Exception in Attribute 1 or until CONDBAT is reached in the system.

The second row indicates that DB2 monitors the number of connections satisfying the scope that is defined by PROFILEID 2 in SYSIBM.DSN_PROFILE_TABLE. When the number of connections in the DB2 system exceeds 50 that is defined in ATTRIBUTE2 column, a DSNT771I message is generated to the system console (if there is no other DSNT771I message being issued within last 5 minutes) and DB2 continues to service the new connection request as WARNING is defined in the ATTRIBUTE1 column.

The third row indicates that DB2 monitors the period of time that the thread can remain idle that satisfies the scope that is defined by PROFILEID 3 in SYSIBM.DSN_PROFILE_TABLE. When the thread stays idle for more than 5 minutes (300 in second) that is defined in ATTRIBUTE2 column, a DSNT771I message is generated to the system console (if there is no other DSNT771I message being issued within last 5 minutes) and DB2 terminates the thread as EXCEPTION_DIAGLEVEL1 is defined in the ATTRIBUTE1 column.

Next, you need to load or reload the profile tables into memory by issuing the following command (which has no parameters):

-START PROFILE

Any rows with a "Y" in the PROFILE_ENABLED column in SYSIBM.DSN_PROFILE_TABLE are now in effect. DB2 monitors any system activities related to threads and connections that meet the specified criteria. When threshold is reached, DB2 takes certain action according to the value specified in the ATTRIBUTE1 column, EXCEPTION or WARNING. See the section of the changes to SYSIBM.DSN_PROFILE_ATTRIBUTES for specific action that DB2 is taking. It is important to note that when you start the START PROFILE command, DB2 begins to monitor the next thread or the next connection activities for the monitoring functions that are in effect.

When you finish monitoring these system related threads and connections activities, stop the monitoring process by performing one of the following tasks:

- ▶ To disable the monitoring function for a specific profile Delete that row from SYSIBM.DSN_PROFILE_TABLE, or change the PROFILE_ENABLED column value to N. Then, reload the profile table by issuing the command START PROFILE.
- ▶ To disable all monitoring for both system related and statement related monitoring that are specified in the profile tables, issue the following command:
- STOP PROFILE

DB2 provides the following commands to manage performance profiles:

▶ **START PROFILE**

Used to load or reload active profiles into memory. This command can be issued from the z/OS console through a batch job or IFI. This command can also be used by tools, such as Optim Query Tuner.

▶ **DISPLAY PROFILE**

Allows you to see if profiling is active or inactive. This command can be issued from the z/OS console, using a batch job or instrumentation facility interface (IFI). This command can also be used by tools, such as Optim Query Tuner.

▶ **STOP PROFILE**

Used to stop or disable the profile monitoring function. This command can be issued from the z/OS console, using a batch job or IFI. This command can also be used by tools, such as Optim Query Tuner.

By using profiles, you can easily monitor system level activities. Profiling provides granularity of control for warning and exception handling based on filtering capability for Idle threads, number of connections and number of threads. Now thread and connection resources can be monitored and controlled by filters to correspond to business priorities.

The information collected in these table was used by tools such as the Optimization Service Center and Optimization Expert (see *IBM DB2 9 for z/OS: New Tools for Query Optimization*, SG24-7421), and used by their follow on tools, Optim Query Tuner and Optim Query Workload Tuner for DB2 for z/OS.

12.3 OMEGAMON PE Extended Insight

OMEGAMON PE Extended Insight is an advanced way to monitor the database workload (SQL) of your applications and solutions. Extended Insight is made available as a feature of the OMEGAMON Performance Expert offering only. This solution allows you to do the following tasks:

- ▶ Manage response time SLAs, Monitor application health
- ▶ Identify the problem workload (user, client machine, application, and so on)
- ▶ Identify the problem period
- ▶ Identify the problem SQL
- ▶ Identify the problem layer

OMEGAMON PE Extended Insight is a database workload monitoring based on end-to-end transaction response times that offers the following capabilities:

- ▶ Get total response times and response time breakdown (application, driver, network, data server) per defined workload/cluster (for example per system, application, user)
- ▶ Compare workload from various servers / applications
- ▶ Select a time period for analysis
- ▶ Get top SQL statements per defined workload
- ▶ Identify top clients contributing in the workload

OMEGAMON PE EI (Extended Insight) takes advantage of the DB2 10 monitoring enhancements to help solve one of the most critical questions in monitoring performance of distributed applications, that is “where is the application spending its time?”. Today’s distributed applications are deployed in complex architectures involving many components. To be able to identify which piece of the infrastructure is responsible for an elongated response time is very important.

12.3.1 Examples

Figure 12-8 illustrates the components of the End-to-End response time of a transaction. It shows that, for this example, OMEGAMON PE EI can provide information about the time spent in the path **Application - Application Server - JCC driver - Network - Database - OS** individually and per layer.

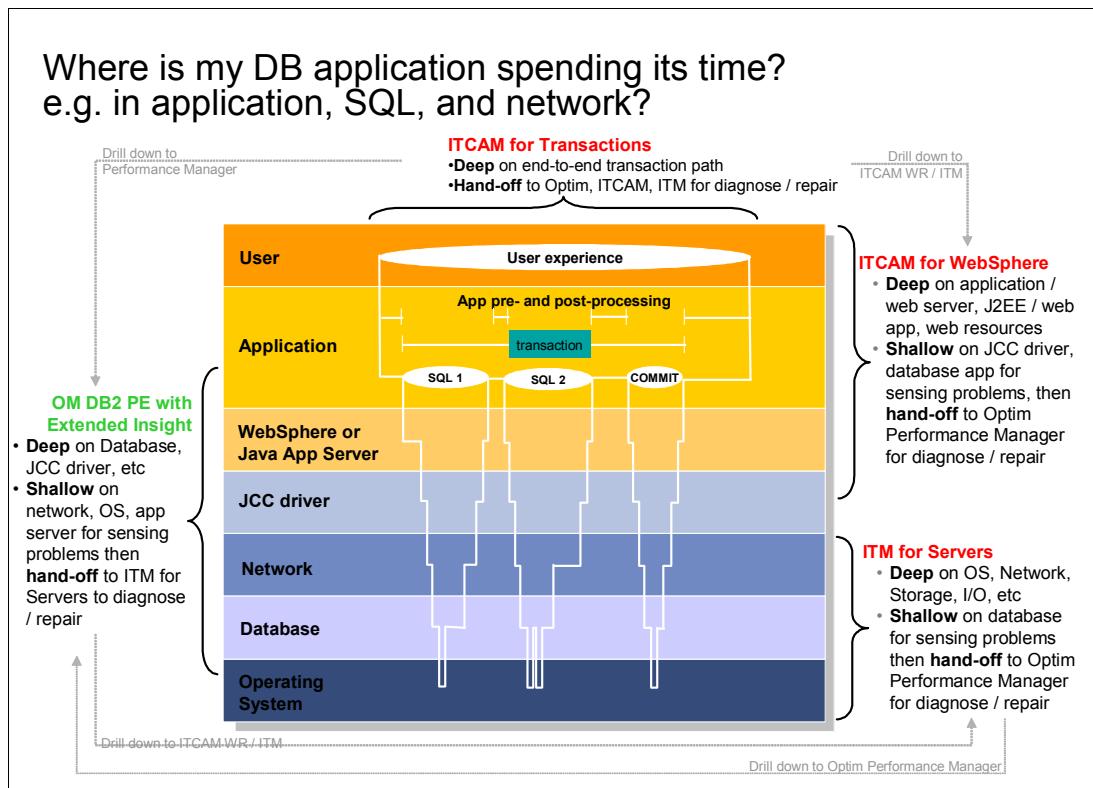


Figure 12-8 Components of an End-to-End response time

For more details on installing and getting started with OMEGAMON PE EI, see the IBM developerWorks® tutorial *Get started with DB2 Performance Expert Extended Insight Feature* at

<http://www.ibm.com/developerworks/data/tutorials/dm-0906db2expertinsight1/>

At a glance, the prerequisite steps include:

- OMEGAMON Performance Expert needs to be installed in z/OS.
- The Extended Insight feature needs to be activated.
- Installation of OMEGAMON Performance Manager is required.
- DB2 Server is required as a repository for OMEGAMON Performance Manager.
- OMEGAMON Performance Expert Fixpack 1 is **mandatory** for supporting EI in DB2 for z/OS.

For information about hardware and software requirements for installing Optim Performance Manager 4.1, see the following website:

http://publib.boulder.ibm.com/infocenter/idm/v2r2/topic/com.ibm.datatools.perfmgmt.installconfig.doc/pm_install_reqs.html

For instructions about how to install the product, see the following website:

http://publib.boulder.ibm.com/infocenter/idm/v2r2/topic/com.ibm.datatools.perfmgmt.installconfig.doc/pm_installconfigure.html

Important: Optim Performance Manager 4.1.0.1. It is Fixpack 1 of Optim Performance Manager, and is mandatory for supporting DB2 for z/OS. You can get Fixpack 1 from:

<http://www.ibm.com/support/docview.wss?rs=434&uid=swg27008647#opm-1ib>

You can find the release notes at this location:

http://publib.boulder.ibm.com/infocenter/idm/docv3/index.jsp?topic=/com.ibm.datatools.perfmgmt.relnotes.doc/pm_4101_release_notes.html

Figure 12-9 shows the architectural overview of OMEGAMON PE and the EI feature.

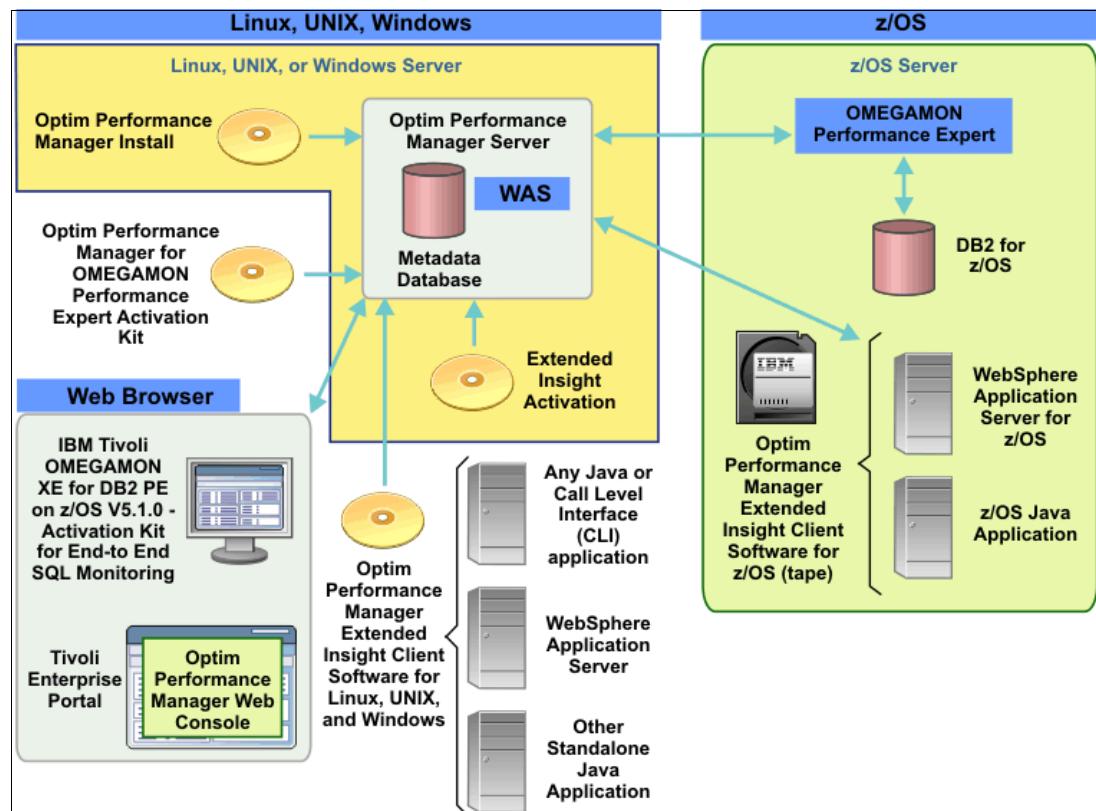


Figure 12-9 Overview of Optim Performance Expert Extended Insight architecture

During the project for this book, we installed this solution from scratch and we found no major inconveniences in the process. Notice that the deployment of the tool involves distributed servers as well, this is not a z/OS implementation only. We tested the Optim Performance Manager installed in a Windows and in a zLinux server with success. You might need to plan the steps and multidisciplinary tasks carefully in advance in order to assure a smooth installation.

To get an idea of the user interface, see Figure 12-10, which shows an example of the Extended Insight Analysis Dashboard for one of our test systems.

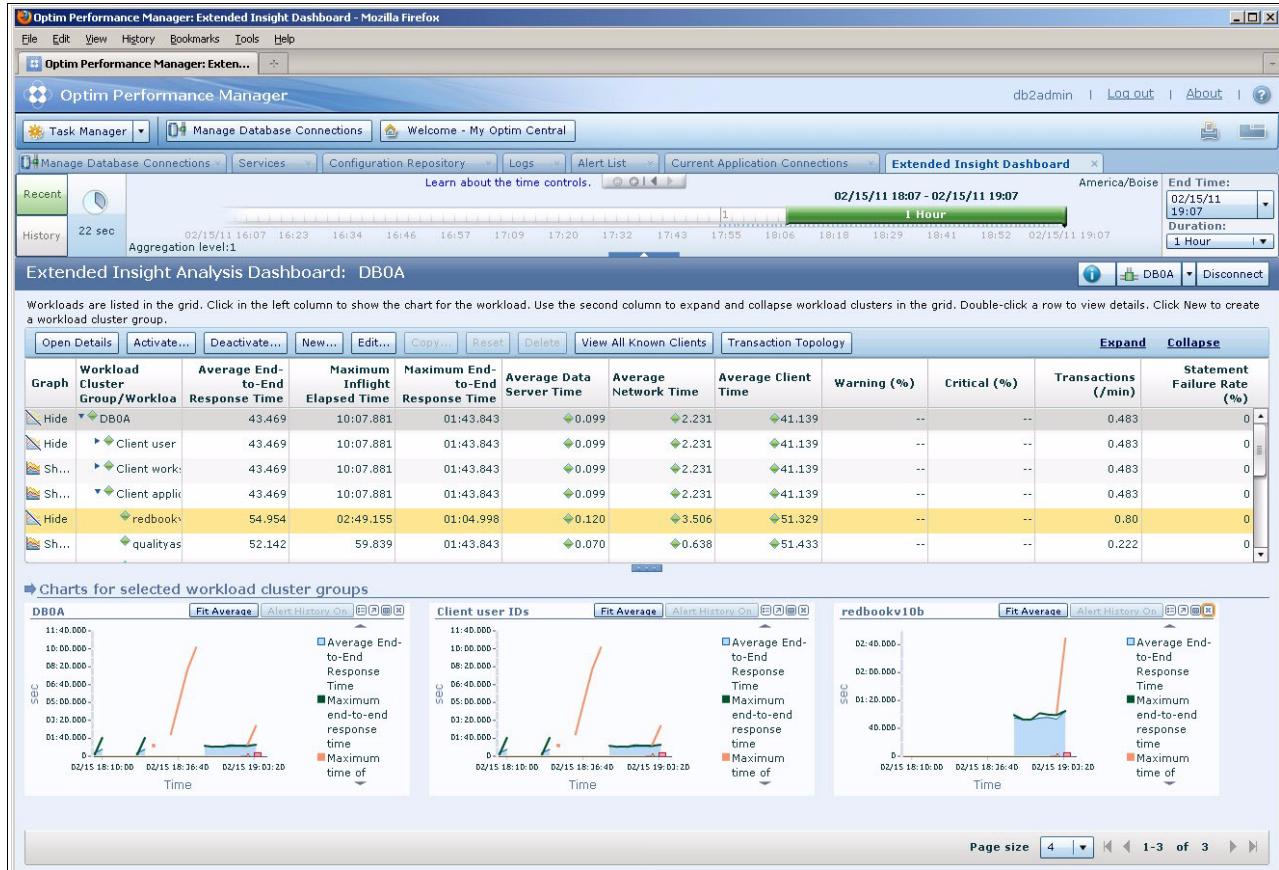


Figure 12-10 OMEGAMON PE EI Analysis Dashboard

Figure 12-11 shows the product's main screen. From this panel you can administer monitored databases by selecting the **Manage Database Connections** option, as shown in the figure.

In this section we intend to offer you a glance at the procedure involved in getting OMEGAMON PE EI working against a DB2 10 for z/OS database. It is not meant to be a guide to product installation and configuration. For more information, see these sources:

- ▶ The IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS website:
<http://www.ibm.com/software/tivoli/products/omegamon-xe-db2-peex-zos/features.html>
- ▶ The IBM Tivoli OMEGAMON XE for DB2 Performance Expert publications:
http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/index.jsp?topic=/com.ibm.omegamon.xe_db2.doc/ko2mee1063.htm
- ▶ The technote “Deployment Alternatives - OMEGAMON PE for DB2 on z/OS Extended Insight Analysis Dashboard” available at:
<http://www.ibm.com/support/docview.wss?uid=swg21456995&myns=swgtiv&mynp=OCSSUSP&mync=R>

Follow these steps:

1. On the main screen (Figure 12-11), select the **Manage Database Connections** option.

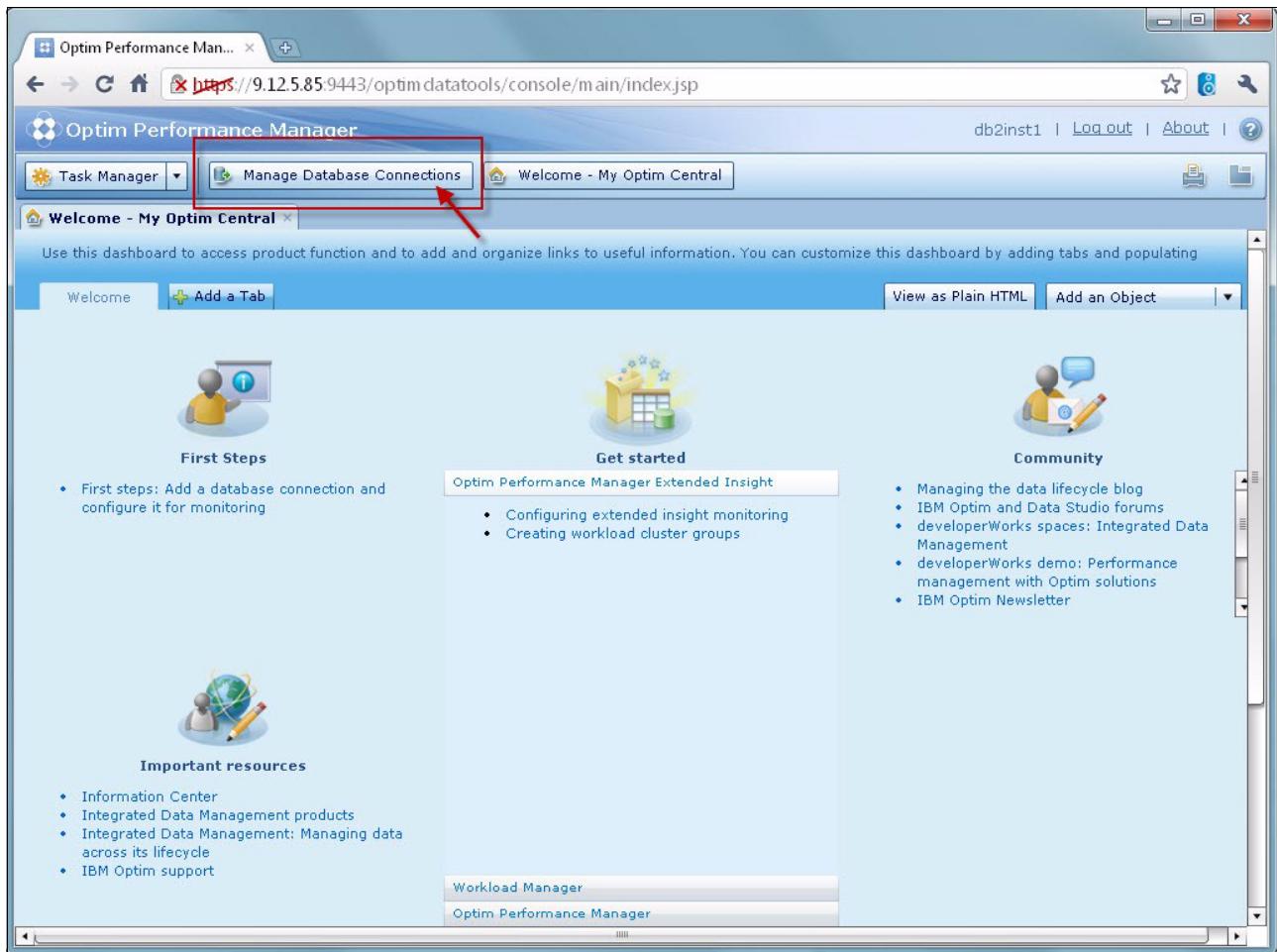


Figure 12-11 OMEGAMON Performance Manager main screen

2. Move to the next panel (Figure 12-12) where you can add a database to be monitored. Here we show an example of how we set the systems in our test environment.

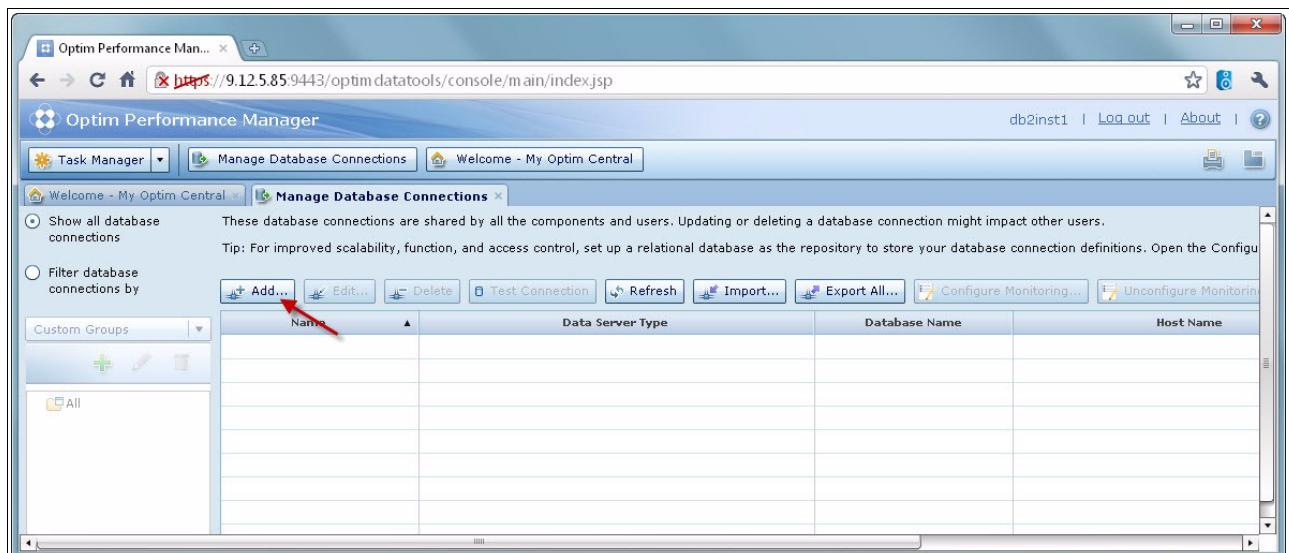


Figure 12-12 Adding a database for monitoring to OMEGAMON PE EI

3. In this panel, enter the connectivity information of the target DB2 for z/OS database, such as IP or DNS entry, port and authorization information. You need to enter the OMEGAMON PE port number and IP or DNS entry. Figure 12-13 shows an example.

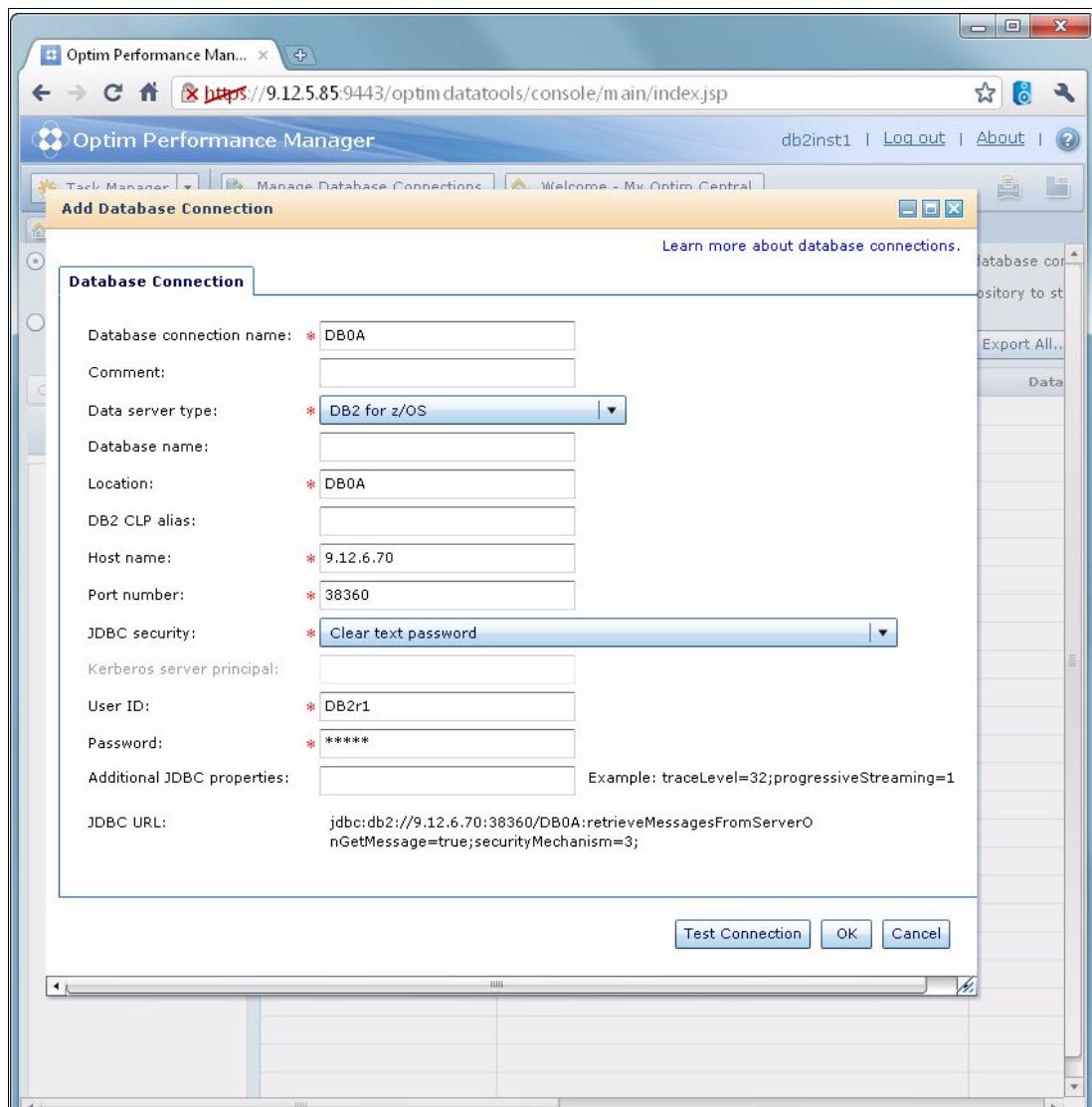


Figure 12-13 OMEGAMON PE EI - configuring a database

4. After testing and verifying connectivity by the option **Test Connection** of this panel, move forward and select the option **Configure Monitoring**, as shown in Figure 12-14.

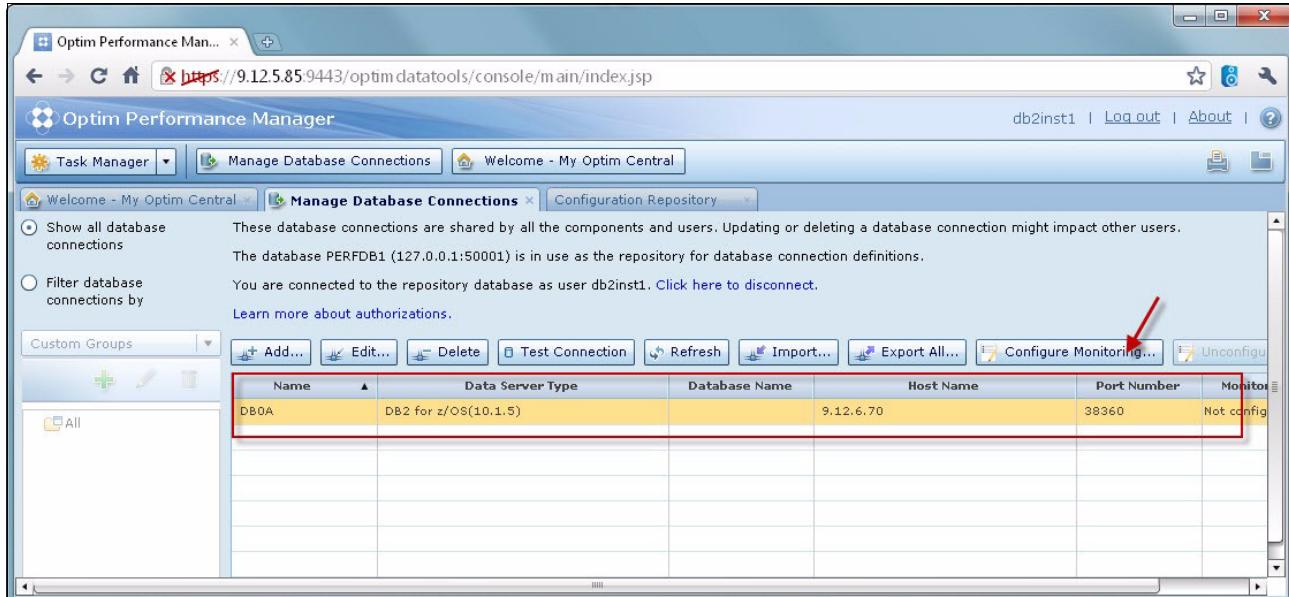


Figure 12-14 OMEGAMON PE EI configure monitoring

Figure 12-15 shows a section of the **Configure Monitoring** process. This panel reminds you that this tool relies on the IFCIDs 316, 317, and 318 being active in the DB2 for z/OS subsystem.

The screenshot shows a web-based interface for the Optin Performance Manager. The title bar reads "Optin Performance Man..." and the URL is "https://9.12.5.85:9443/optimdatatools/console/main/index.jsp". The main content area is titled "Step 3 of 5: View the configuration settings for DB2 and OMEGAMON Performance Expert".
Settings for metric collection:
Dynamic SQL text: YES ⚠
Required OMEGAMON Performance Expert collector configuration: Activate the collection of IFCID 316,317,318 using the appropriate START TRACE command. Required OMEGAMON Performance Expert collector configuration: Activate the collection of IFCID 316,317,318 using the appropriate START TRACE command.
Dynamic SQL metrics: NO
Static SQL metrics: NO
Extended Insight Monitoring:
Extended Insight Monitoring: ON ⚠
OMEGAMON Performance Expert is enabled for Extended Insight monitoring: The test was successful.
Static SQL texts are read from system catalog: YES ⚠

For details, see the following publication:
[IBM Tivoli OMEGAMON XE for DB2 Performance Expert and Performance Monitor on z/OS: Configuration and Customization](#)
Buttons at the bottom include "Copy to Clipboard", "Print", "Previous", "Next", "Finish", and "Cancel".

Figure 12-15 Review configuration settings for OMEGAMON PE EI

12.3.2 Configuring a CLI application

In order to monitor an application end-to-end, you need to change the connectivity setting at the client side. This section shows an example of configuring a CLI application. See the OMEGAMON PE EI documentation for more details for other connectivity options.

OMEGAMON PE EI provides a Configuration Tool to assist with this process. Figure 12-16 shows the main screen of this application. After the welcome screen, you receive a window that allows you to input which type of communication to configure. In this example, we configured all the available options:

- ▶ CLI applications
- ▶ JDBC applications
- ▶ WebSphere applications

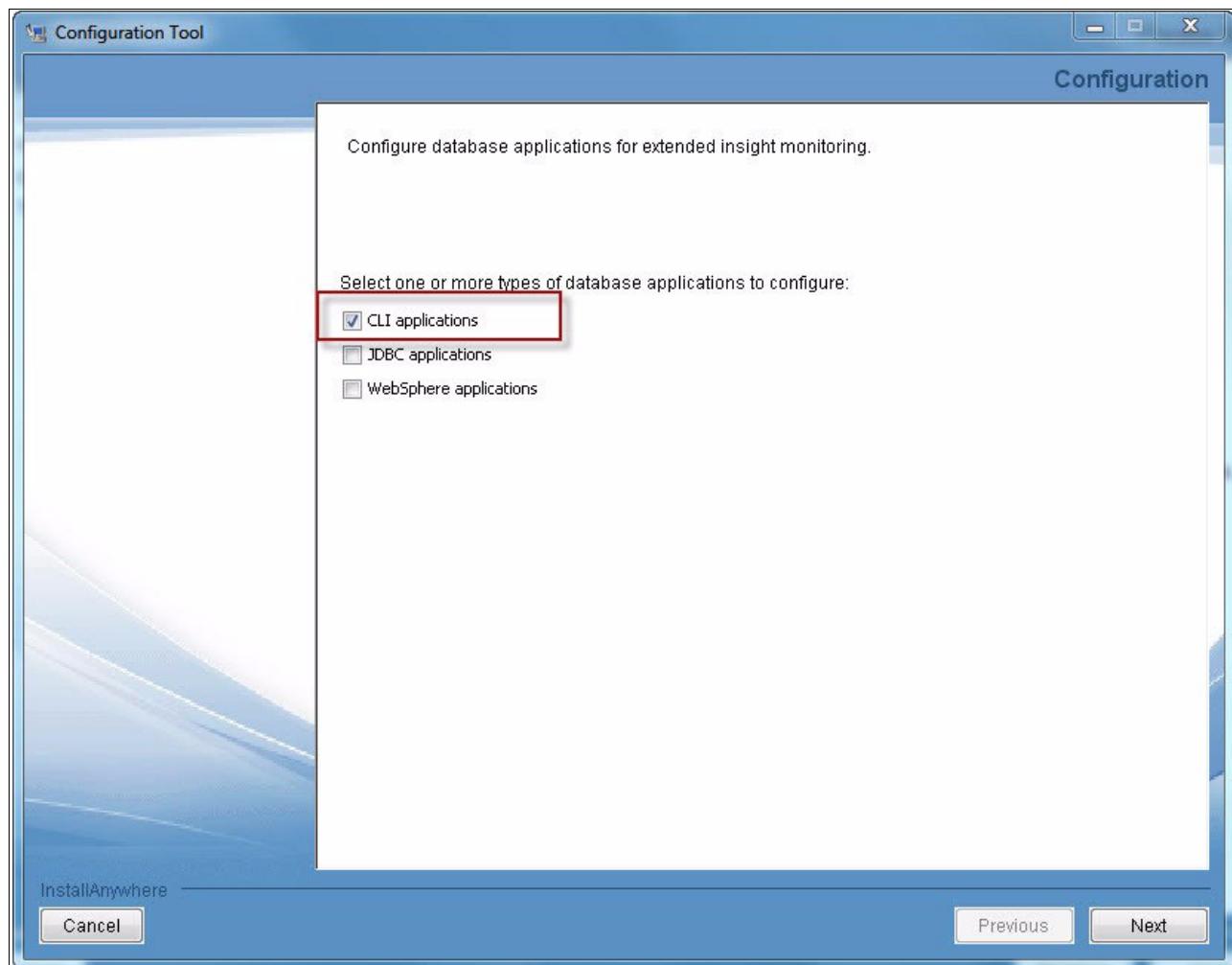


Figure 12-16 OMEGAMON PE EI Client configuration tool

The configuration file db2dsdriver.cfg contains database directory information and client configuration parameters in a human-readable format that are used by DB2 Data Servers and Clients. See “DB2 9 for z/OS: Distributed Functions, SG24-6952-01 for more and information about DB2 Drivers and the db2dsdriver.cfg file.

The next screen is used for indicating which **db2dsdriver.cfg** file to update. Figure 12-17 shows an example. Notice that the file location can vary depending on your actual installation.

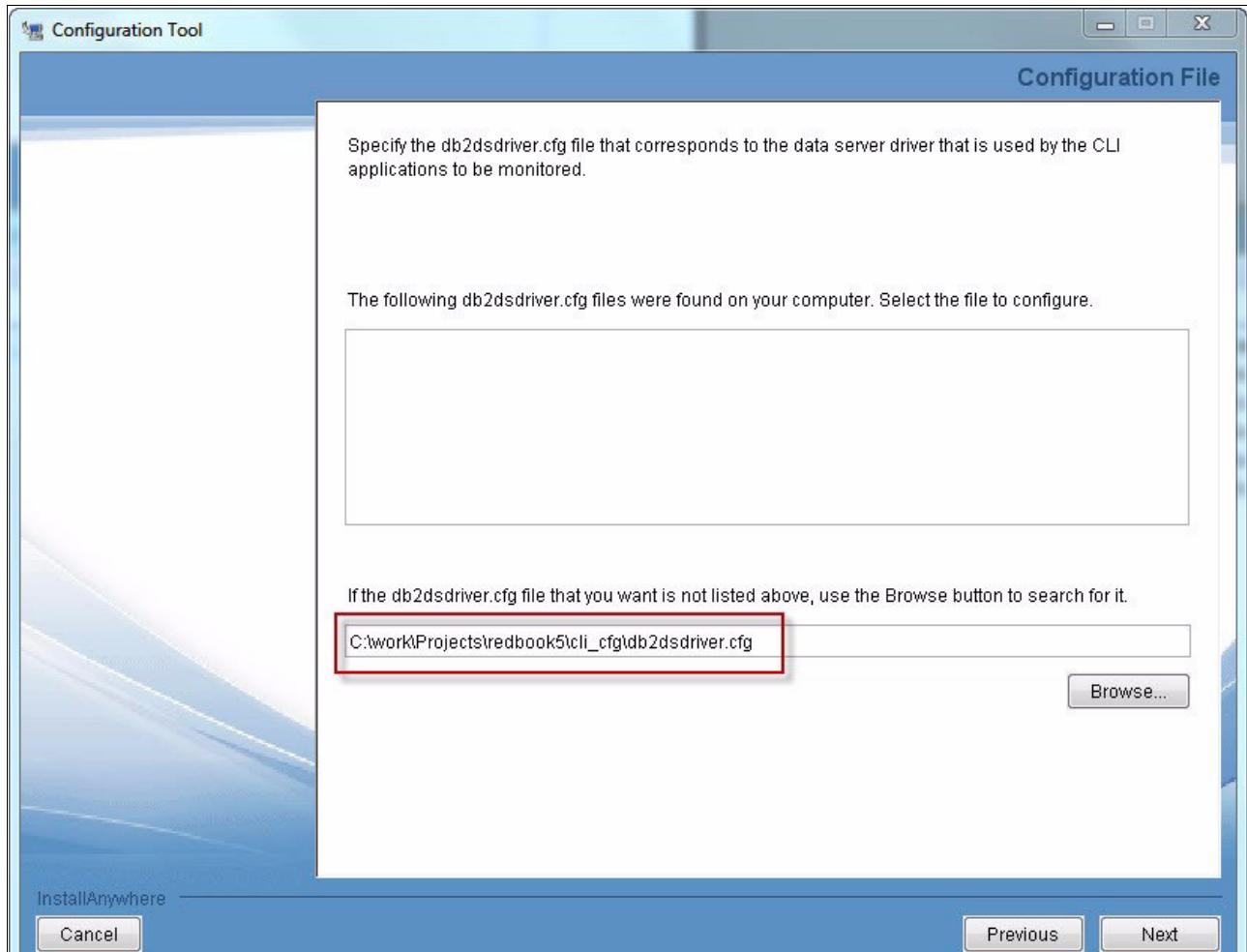


Figure 12-17 Indicating the db2dsdriver.cfg file to update for OMEGAMON PE EI

The following screen will ask for the IP address or DNS entry for the OMEGAMON Performance Manager server, that is a distributed server and not DB2 for z/OS, and its port.

Figure 12-18 illustrates the final screen of this tool. It contains a summary of the information entered and the application performs a checkup of these parameters when you hit the **Configure** button.

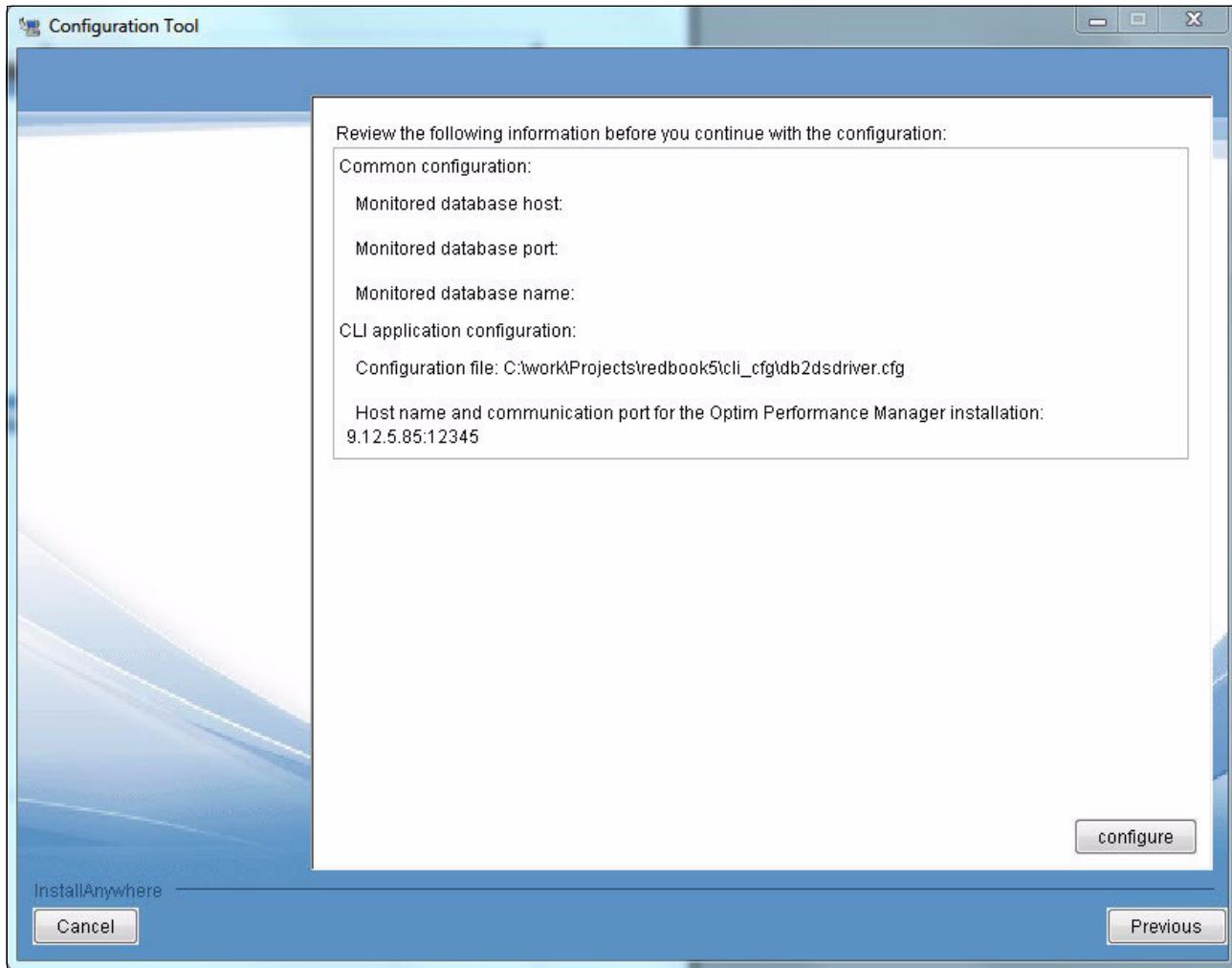


Figure 12-18 Configuration of a CLI application for OMEGAMON PE EI

Example 12-5 shows the modified db2dsdriver.cfg file as created by this process.

Example 12-5 db2dsdriver.cfg example for OMEGAMON PE EI support

```
<?xml version="1.0" encoding="UTF-8"?><configuration>
  <DSN_Collection>
    <dsn alias="alias1" host="server1.net1.com" name="name1" port="50001"/>
    <!-- Long aliases are supported -->
    <dsn alias="longaliasname2" host="server2.net1.com" name="name2" port="55551">
      <parameter name="Authentication" value="Client"/>
    </dsn>
  </DSN_Collection>
  <databases>
    <database host="server1.net1.com" name="name1" port="50001">
      <parameter name="CurrentSchema" value="OWNER1"/>
      <WLB>
        <parameter name="enableWLB" value="true"/>
        <parameter name="maxTransports" value="50"/>
      </WLB>
      <ACR>
        <parameter name="enableACR" value="true"/>
      </ACR>
    </database>
    <!-- Local IPC connection -->
    <database host="localhost" name="name3" port="0">
      <parameter name="IPCInstance" value="DB2"/>
      <parameter name="CommProtocol" value="IPC"/>
    </database>
  </databases>
  <parameters>
    <parameter name="GlobalParam" value="Value"/>
    <parameter name="connectionSupervisorProperties"
value="controllerURL=9.12.5.85:12345,dataSourceLookupInterval=20"/>
    <parameter name="connectionSupervisorLibrary" value="C:\Program Files
(x86)\IBM\Optim_pureQueryRuntime\pureQuery\bin\pqcmx"/>
  </parameters>
</configuration>
```



A

Recent maintenance

With a new version of DB2 reaching general availability, the maintenance stream becomes extremely important. Feedback from early users and development of additional functions cause a flux of APARs that enrich and improve the product code.

In this appendix, we look at recent maintenance for DB2 10 for z/OS that generally relates to performance:

- ▶ DB2 APARs
- ▶ z/OS APARs
- ▶ OMEGAMON/PE APARs

These lists of APARs represent a snapshot of the current maintenance at the moment of writing. As such, the list becomes incomplete or even incorrect at the time of reading. Make sure that you contact your IBM Service Representative for the most current maintenance at the time of your installation. Also check on IBM RETAIN® for the applicability of these APARs to your environment, as well as to verify pre- and post-requisites.

Use the Consolidated Service Test (CST) as the base for service as described at this website:
<http://www.ibm.com/systems/z/os/zos/support/servicetest/>

The current quarterly RSU is CST4Q12 (RSU1212), dated January 3, 2013 for DB2 9 and DB2 10. It contains all service through the end of September 2012 not already marked RSU, PE resolution, and HIPER / Security / Integrity / Pervasive PTFs, and their associated requisites and supersedes through the end of November 2012. It is described at this website:
<http://www.ibm.com/systems/resources/RSU1212.pdf>

A.1 DB2 APARs

In Table A-1 we present a list of APARs providing functional and performance enhancements to DB2 10 for z/OS.

This list is not and cannot be exhaustive; check RETAIN and the DB2 website.

Table A-1 DB2 10 current function and performance related APARs

APAR #	Area	Text	PTF and notes
II10817	Storage	Info APAR for storage usage error	
II14219	zIIP	zIIP exploitation <i>support use</i> information	
II14334	LOBs	Info APAR to link together all the LOB support delivery APARs	
II14401	Migration	Info APAR to link together all the migration APARs	
II14426	XML	Info APAR to link together all the XML support delivery APARs	
II14441	Incorrout PTFs	Preferred DB2 9 SQL INCORROUT PTFs	
II14464	Migration	DB2 V8 migration/fallback info APAR to/from DB2 9 (continued from II14401)	
II14474	Migration	Prerequisites for migration to DB2 10 from V8	
II14477	Migration	Prerequisites for migration to DB2 10 from DB2 9	
II14564	Migration	Continuation of II14477	
II14587	Workfile	DB2 9 and 10 workfile recommendations	
II14619	Migration	Info APAR for DB2 10 DDF migration	
PK28627	DCLGEN	Additional DCLGEN option DCLBIT to generate declared SQL variables for columns defined as FOR BIT DATA (COBOL, PL/I, C, and C++).	UK37397
PK76100	Star join (EN_PJSJ)	Enable dynamic index ANDing for star join (pair wise)	UK44120 also V9
PK83397	Utilities	New DSNZPARM REORG_IGNORE_FREESPACE	UK50932 V9 only
PK92339	DDF	PRIVATE_PROTOCOL new DSNZPARM	UK51679 for V9 also V8
PM00068	DSMAX	Support up to 100000 open data sets in the DB2 DBM1 address space	UK58204/5 V8 and V9
PM01821	Migration	Conversion of DBRMs to packages	UK53480/1 also V8 and V9
PM04968	Premigration	This APAR adds the DB2 10 DSNTIJPM job to DB2 V8 and DB2 9 under the name DSNTIJPA	V8 (UK56305) and V9 (UK56306)
PM13466	Pricing	Use the IFAUSAGE FBFE keyword when SMF 89 detailed data collection is enabled	UK62326/7 also V8 and V9
PM13467	Pricing	Use the IFAUSAGE FBFE keyword when SMF 89 detailed data collection is enabled	UK62328/9 also V8 and V9

APAR #	Area	Text	PTF and notes
PM13525	SQL procedures	Implicit auto regeneration for native SQL procedures (BIND PACKAGE DEPLOY or REBIND PACKAGE or CALL statement for a native SQL procedure)	UK67267 also V9
PM13631	IX on expression	Issues when index on expression is created or regenerated in V10 CM9.	UK68476 for V9
PM17336	Work file	Modify DSNWTFG to also generate work file table spaces with SECQTY of 0.	UK69607 also V9
PM17542	Open/close	Enable new z/OS 1.12 allocation interface.	UK60887/8 also V8 and V9
PM17665	DDF	Changes to authorization checking at a DB2 for z/OS server when Private Protocol is disabled.	UK65969 also V8 and V9
PM18196	DFSORT	Optimization of DFSORT algorithms for selecting the sort technique that makes the best use of available storage	UK62201
PM18557	Restart	DB2 to support z/OS R12 changes to improve high allocation requests processing.	UK59887/8 also V8 and V9
PM19034	DSNZPARM	CHECK_FASTREPLICATION parameter to control FASTREPLICATION keyword on DSSCOPY command of CHECK utilities.	UK63215 also V8 and V9
PM19584	LOAD	LOAD utility performance improvement option for data that is presorted in clustering key order and FORMAT INTERNAL support for LOAD and UNLOAD.	UK68097 also V9
PM21277	DB2 utilities	REORG, CHECK INDEX, and REBUILD INDEX users who use DB2 Sort for z/OS.	UK61213 also V8 and V9
PM21747	DB2 Sort Tool	Performance enhancements	UK60466
PM24721	BIND	BIND performance improvement when using LOBs in DB2 catalog. Also RTS fix.	UK63457
PM24723	IFCID 225	Provide real storage value with z/OS support (see also OA35885 and PM37647)	UK68652
PM24808	DB2 installation	Several installation changes	UK63971 also V9
PM24937	Optimizer	Various fixes for optimization hints	UK66087
PM25271	IRLM	IRLM Large Notify support	UK66475
PM25282	IRLM	IRLM Large Notify support. IRLM uses ECSA storage to handle response data to a NOTIFY request. Prior to DB2 10, the amount of data per member was 4 MB. In DB2 10, this limit has been increased to 64 MB. IRLM NOTIFY RESPONSE exceeding 4 MB is processed using IRLM private storage.	UK64370
PM25357	Optimizer	Getpage increase using subquery	UK63087
PM25525	REORG	PARALLEL keyword on REORG with LISTDEF PARTLEVEL (apply with PM28654/UK64589)	UK64588 also V9
PM25635	ADMIN_INFO_SQL	Corrected DDL and STATS issues (also for DSNADMSB)	UK62150

APAR #	Area	Text	PTF and notes
PM25648	REORG	Improving REORG concurrency on defer ALTER materialization.	UK70310
PM25652	DSNACCOX	Enhancement to advise REORG on Hash Access objects based on overflow index ratio versus total rows	UK66610
PM25679	Optimizer	Enhancement for APREUSE/APCOMPARE	UK70233
PM26475	Modeling production	New DSNZPARMs and profile monitoring keywords values have been added to allow modelling of CPU speed, number of processors, sort pool, RID pool, and buffer pool settings. These new attributes are only used when determining an access path and are not used elsewhere. The actual values for the modelled settings remain unchanged. See also PM26973.	UK65332 also V9, V8
PM26480	DDF	Availability (MODIFY DDF ALIAS...) new functions	UK63820
PM26762	DSNZPARM	FLASHCOPY_PPRC (DB2 10 only) and REC_FASTREPLICATION for use by utilities	UK63366 also V8, V9
PM26781	DDF	Availability (MODIFY DDF ALIAS...) preconditioning	UK63818
PM26977	Security	Separate system privileges from system DBADM authority	UK65205
PM26973	Modeling production	Further functions.	UK67292 also V9, V8
PM27073	SPT01	Inline LOB preconditioning	UK65379
PM27097	WLM	Support to start and maintain a minimum number of WLM stored procedure address spaces	UK65379
PM27099	LISTDEF	Empty lists change from RC8 to RC4. Important because the new LISTDEF DEFINED keyword defaults to YES	UK64424
PM27811	SPT01	Inline LOB	UK66379
PM27828	UPDATE	UTS update with small record goes to overflow unnecessarily	UK64389
PM27835	Security	DB2 now supports a TCB level ACEE for the authid used in an IMS transaction that calls DB2.	UK70647 also V9
PM27872	SMF	Decompression routine DSNTSMFD and sample JCL DSNTEJDS to call DSNTSMFD	UK64597
PM27962	LOAD	The new LOAD option, INDEXDEFER NPI/ALL indicates to LOAD that the specified index types will be deferred. When the option is specified, the sort and build of the indexes will be skipped during the LOAD and the deferred indexes will be placed in RBDP state. See also PM42560.	UK71419 also V9
PM27973	Segmented TS	Better use free space for SEGMENTED pagesets including UTS PBG and PBR	UK65632
PM28100	Stored procedures	Support JCC JDBC 4.0 driver for Java stored procedures	UK65385
PM28296	Security	Support for secure audit policy trace start	UK65951

APAR #	Area	Text	PTF and notes
PM28385	XML	XML fixes including XMLMODIFY performance	UK66136
PM28458	Casting	Timestamp with timezone, add restrictions for extended implicit cast for set operators	UK63890
PM28500	System profile	Filters on client information fields	UK68364
PM28543	Security	Implicit system privileges have to be separated from system DBADM authority	UK65253
PM28796	SYSROUTINEAUTH	Inefficient access to DB2 catalog during GRANT stored procedures	UK65637
PM28925	Data sharing	New DB2 subsystem parameter in DSN6SYSP DEL_CFSTRUCTS_ON_RESTART	UK66376
PM29037	LOBs	Altered LOB inline length materialization by REORG SHRLEVEL CHANGE.	UK70302
PM29124	CHAR incompatibility	Help with handling the release incompatible change for the CHAR(decimal) built-in function on DB2 10	UK67578
PM29900	Built-in functions	Additions	UK66476
PM29901	Built-in functions	Additions	UK66046
PM30394	Security	DBADM authorities enhancements	UK67132
PM30425	Optimizer	Optimization hints enhancements	UK67637 also V9
PM30468	zIIP	Prefetch and deferred write CPU, when running on a zIIP processor, is to be reported by WLM under the DBM1 address space, not under the MSTR	UK64423
PM30991	RECOVER	Prohibit RECOVER BACKOUT YES after mass delete on segmented or UTS, see also PM52724.	UK66327
PM31003	Data sharing	DELETE data sharing member	UK65750
PM31004 PM31006 PM31009	Data sharing	DELETE data sharing member	UK67512 UK67958 UK69286
PM31214	Hash	HASH LOAD performance	Closed, moved to next release
PM31243	REORG	REORG FORCE to internally behave like CANCEL THREAD	Closed as SUG
PM31313	Temporal	ALTER ADD COLUMN to propagate to History Tables.	UK70215
PM31314	Temporal	TIMESTAMP WITH TIMEZONE.	UK71412
PM31614	Packages	Improvement in package allocation.	UK66374
PM31641	LOGAPSTG	Default changed for Fast Log Apply from 100 to 500 MB	UK66964
PM31807	IRLM	IRLM support for DB2 DSNZPARM DEL_CFSTRUCTS_ON_RESTART (PM28925) to delete IRLM lock structure on group restarts.	UK65920

APAR #	Area	Text	PTF and notes
PM31813	DSNZPARM	New DSNZPARM DISABLE_EDMRTS to specify whether to disable collection of real time statistics (RTS) by the DB2 Environmental Descriptor Manager (EDM). This system configuration parameter requires PM37672 to be fully enabled.	UK69055
PM33501	DSNZPARM	Add a DSNZPARM to disable implicit DBRM to package conversion during BIND PLAN with MEMBER option and automatic REBIND processing.	UK68743
PM33767	Optimizer	Various enhancements and fixes (OPTHINT, APRETAINUP, EXPLAIN PACKAGE).	UK69377
PM33991	Migration	Several installation jobs fixes.	UK69735 also V8 and V9
PM35190	Catalog	Enable SELECT from SYSLGRNX and SYSUTILX (see also PM42331).	UK73478
PM35284	LOAD	Companion APAR to PM19584 LOAD/UNLOAD FORMAT INTERNAL and LOAD PRESORTED.	UK68098 also V9
PM36177	DSNZPARM	Pre-conditioning APAR that includes changes to support the enhancement for IRLM timeout value for DDL statements enabled by APAR PM37660.	UK69029
PM37112	REORG	Enhancement for log apply phase of REORG SHRLEVEL CHANGE when run at partition level (see also PM45810).	UK71128 Also V9
PM37293	DSNZPARM	New DB2 zparm REORG_LIST_PROCESSING to specify the default setting of the PARALLEL option (see PM25525) when deciding whether REORG would process the LISTDEF partitions in parallel or serially.	UK69494
PM37300	DDF	Authorization changes when there is no private protocol (see also PM17665 and PM38417). DSN6FAC PRIVATE_PROTOCOL reinstated in V10 with new option AUTH.	UK67639 also V8 and V9
PM37625	DSNZPxxx module	Preconditioning support in DB2 subsystem parameter macro DSNDSPRM for APARs PM24723, PM36177, PM31813, and PM33501.	UK67634
PM37647	Real storage monitoring	External enablement for APAR PM24723 (IFCID 225 REAL STORAGE STATISTICS ENHANCEMENTS	UK68659
PM37660	DSNZPARM	DDLTOX in DSNDSPRM is introduced a separate time out factor for DDL and DCL (GRANT, REVOKE, and LOCK) statements. The time out value is the product of DDLTOX and the IRLM time out value specified by DSNDSPRM.IRLMRWT.	UK69030
PM37672	DSNZPARM	New DISABLE_EDMRTS zparm allows to disable collection of real time statistics by the DB2 Environmental Descriptor Manager.	UK69058

APAR #	Area	Text	PTF and notes
PM37816	DSNZPARM	Follow on to APAR PM33501, it adds DSNZPARM DISALLOW_DEFAULT_COLLID in DSN6SPRM which can prevent using DSN_DEFAULT_COLLID_plan-name on implicitly generated packages during the DB2 automatic DBRM to package conversion process.	UK69199
PM37956	Utilities	New function to trace accounting CPU time usage by DB2 utilities and to determine how much of the utility CPU time is spent in sort processing.	UK70426 also V8 and V9
PM38164	Access path	During access path selection, index probing can have repeated access to SYSINDEXSPACESTATS to retrieve NLEAF. This occurs if NLEAF is NULL. Make index probing more fault tolerant when NLEAF is NULL.	UK71333
PM38417	DDF	Complete DSNZPARM related changes for PM37300 (security with private protocol removal)	UK74175 also V8 and V9
PM39342	Online compression	Build Dictionary routine for compression during INSERT was modified not to be re-driven by subsequent INSERTs if the dictionary could not be built and MSGDSNU235I is issued.	UK68801
PM40388	REORG	EDM DBD SPACE shortage when running multiple COPY and REORG jobs on different members in the same data sharing group. See also PM52727.	UK70513 also V9, V8
PM40501	IFI	The commands -STOP DATABASE, -SET LOG, -ALTER BUFFERPOOL, -SET SYSPARM are allowed to run synchronously, Used by routines like SYSPROC.ADMIN_UPDATE_SYSPARM.	UK69784 also V9
PM41447	DDF security	Resolve issues with CICS or IMS related new user processing at a DB2 10 for z/OS TCP/IP requester system.	UK70483
PM42331	Catalog	Enable SELECT from SYSLGRNX and SYSUTILX.	UK71875
PM42528	Data sharing	Delete data sharing member	UK74381
PM42560	LOAD	Complete INDEXDEFER	UK71420
PM42924	RUNSTATS	Optimize sequential prefetch requests during RUNSTATS TABLESAMPLE	UK70844
PM43293	DDF	For remote connections in data sharing reaching MAXDBAT, allow management of queued connections with new ZPARMs MAXCONQN and MAXCONQW.	UK90325
PM45318	OPEN/CLOSE	Latch 32 contention reduction	UK72557
PM45650	LOBs	RECOVER BACKOUT YES for LOBs, table space set to AUXW	UK77584
PM45810	REORG	Enhancement for log apply phase of REORG SHRLEVEL CHANGE when run at partition level. (See also PM37112)	UK71128

APAR #	Area	Text	PTF and notes
PM48358	ALTER COLUMN	Opaque subsystem parameter RESTRICT_ALT_COL_FOR_DCC can be used to prevent use of the ALTER TABLE ALTER COLUMN with SET DATA TYPE, SET DEFAULT, and DROP DEFAULT when Data Capture Changes is enabled on the target table.	UK74760 also V9
PM49816	MSTR	Apply OA37821. High CPU in DB2 10 MSTR after upgrade to z/OS 1.12	UK74840
PM50140		Mass delete locking with DGTT	UK74556
PM51467	Data sharing	Reduce CF CPU utilization in a DB2 10 for z/OS data sharing environment during Delete_Name processing	UK75324
PM51945	Data sharing	Delete member completion	UK74381
PM52727	REORG	Complete PM40388	UK75499
PM52724	Mass delete	Mass deletes ends up with Lock escalation on SYSCOPY. Follow on to PM30991.	UK80113
PM52788	RTS	Reduction of NOTIFY messages	UK76344
PM55051	REORG	Performance enhancement for REORG TABLESPACE PART with NPSIs	UK78229 also V9
PM55933	IFCID	IFCID 367 is enabled to support XML storage serviceability trace	UK77001 also V9
PM56355	Optimizer	DB2 to favor range list index scan over regular single index scan for a query with data-dependent pagination	UK77343
PM56429	Space	Solve unexpected space growth of Partition By Growth UTS or single table segmented table space with large lock failures.	UK82787 Also V9
PM56542	PREPARE	incorrect use the index probing feature when no search value is available and assume all rows qualify for the predicate or index when there are zero rows in the table.	UK76645
PM56363	DDF	Excessive CPU consumption for DDF when processing authentication related events	UK76060
PM56725	DBM1	Reduce TCB time and open/close	UK77298
PM56845	DSNZPARM	With OPT1ROWBLOCKSORT zPARM, when OPTIMIZE FOR 1 ROW is used with a query, DB2 will disable sort access paths when a no-sort choice is available	UK77500
PM57632	LOAD	LOAD SHRLEVEL CHANGE PARALLEL support	UK78632
PM58915	DSNWMSSGS	Update of DB2 tracing	UK90618 also V9
PM60233	BIND	Unnecessary index probing	UK77918
PM60236	BIND	Index probing performance	UK78390
PM60732	Space manager	Index space map search for free page during index page split takes a long time for large indexes and causes time-outs.	UK78678

APAR #	Area	Text	PTF and notes
PM60826	Workfiles	Disable predicate PROCs to avoid issues	UK79281
PM62709	IRLM	Extra SRBs scheduled	UK80552
PM62797	Accounting	If IFCID3 and IFCID369 are on, DB2 aggregates data for each transaction grouped by connection type externalized in an IFCID369 (every 1 min.) or via an IFCID READS call. A new statistics class 9 includes IFCID369	UK81047
PM63219	Instrumentation	XES false contention on resource lock requests are not accounted for correctly in QTGSFLMG	UK79368
PM63505	RESTORE	Improve performance of RESTORE SYSTEM utility during log apply phase	UK78910 only V9
PM63753	Sort pool	Allow minimum of 240 KB	UK79550
PM64230	Segmented	Segmented table space with insert and mass delete within commit do not reuse the datapages.	UK80107 also V9
PM64226	LOBs in catalog	LOB table spaces of DSNDB01 experience significant space growth from BIND/REBIND, DDL, utility activity such as REORG	UK83215
PM64602	Locking	Performance option for page sets started read only in data sharing	UK79898
PM65236	Storage	Bind option RELEASE(DEALLOCATE) with searched SQL UPDATE or DELETE statement (no cursor) causes storage issues.	UK79701
PM65360	CPU	High DB2 MSTR SRB CPU time during DB2 storage contraction at thread deallocation or commit time.	UK80191
PM65550	Unload/Reorg	UNLOAD and REORG WHEN processing has been changed to use a more optimal path to improve the performance.	UK80678
PM65767	Workfiles	Workfile table space selection algorithm is not selecting workfile PBG table spaces as expected.	UK81340
PM66173	Latching	LC23 and LC32 contention or page latch contention for P-LOCKs due to [pool fragmentation.	UK80522
PM66882	RID	Use V9 in-memory quick RID sort to avoid issues for RIDs already almost sorted.	UK81045
PM70046	DSNZPARM	DB2 9 improved the formula for balancing the costs of input/output and CPU speeds by enabling and setting OPTIOWGT value. DB2 10 has changed the default subsystem parameter OPTIOWGT and the DSN_PROFILE_ATTRIBUTES from DISABLE to ENABLE in order to prevent performance problems when migrating from V8.	UK83168
PM70181	RECOVER	Performance enhancement for the RECOVER with BACKOUT YES utility by avoiding performing periodic commits during LOG UNDO phase.	UK81719
PM70270	Buffer pool	Solve lower buffer hit ratio or more synchronous read I/O if VPSEQT is set to a small value.	UK82555 Also V9

APAR #	Area	Text	PTF and notes
PM7114	DB2 provided stored procedure	SYSPROC.ADMIN_UPDATE_SYSPARM stored procedure changes the value of DSNZPARMs.	UK83171 also V9
PM72997	COPY	Avoid increase in the amount of log data for incremental Image Copy especially for Member Cluster table space.	UK83533
PM74659	SPT01	Unusual growth of DSNDB01.SPT01 is encountered in DB2 10	OPEN
PM76924	Diagnosis Guide	Updated versions	UK90197 Also V9

A.2 z/OS APARs

In Table A-2 we present a list of APARs providing additional enhancements for z/OS.

This list is not and cannot be exhaustive; check RETAIN and the DB2 website.

Table A-2 z/OS DB2-related APARs

APAR #	Area	Text	PTF and notes
OA03148	RRS exit	RRS support.	UA07148
OA23049	RSM	PAGEFIX performance.	UA38770
OA31116	1 MB page	Large frames support.	UA57254
OA32599	CPU	High MSTR pre-empt SRB CPU usage in V10 vs. V8.	UA56642
OA32612	OPEN/CLOSE	Reduce Catalog searches through DSAB chains.	UA56302
OA33106	ECSA memory for SRB	Reduce SRB storage usage.	UA56174
OA33529	1 MB page	Large frames support.	UA57243
OA33633	GRS ENQ	SYSZTIOT ENQ improvements in support of DB2 100K data sets.	UA56176
OA33702	1 MB page	Large frames support.	UA57704
OA34865	SMF	Remove cause for accumulation of storage use in subpool 245 (SQA/ESQA).	UA55970
OA35057	Media manager	Important media manager fixes.	UA58937
OA35357	ARM	New CLEANUP_TIMEOUT parameter to determine how long ARM will wait for member cleanup for the terminated system to complete before performing cross-system restart. (see also OA37940)	UA60368
OA35885	RSM	RSM IARV64 macro provides an interface to obtain the real frame and auxiliary storage in use to support an input high virtual storage range.	UA60823
OA36354	Device manager	XTIOT use improved (DB2 CREATE INDEX)	UK61694
OA37697	Device manager	To address long-term the DADSM SCRATCH (DB2 DROP IX) performance with large number of allocated data sets.	UA65808

APAR #	Area	Text	PTF and notes
OA37821	RSM	High DB2 Master CPU noticed for serialization latching on idle system where multiple DB2 10 are active.	UK74840
OA38243	1 MB page	1 MB frame coalesce issue.	UA65482
OA39087	zHPF	DB2 Utility LOAD RESUME with output to a zHPF enabled device.	UA64823

A.3 OMEGAMON PE APARs

In Table A-3 we present a list of APARs providing additional enhancements for IBM Tivoli OMEGAMON XE for DB2 PE on z/OS V5.1.0, PID 5655-W37.

This list is not and cannot be exhaustive; check RETAIN and the DB2 tools website.

Table A-3 OMEGAMON PE GA and DB2 10 related APARs

APAR #	Area	Text	PTF and notes
II14438		Info APAR for known issues causing high CPU utilization.	
PM22628		Various fixes and improvements to be considered part of the GA installation package.	UK61094
PM23887		Various fixes and improvements to be considered part of the GA installation package.	UK61093
PM23888		Various fixes and improvements to be considered part of the GA installation package.	UK61142
PM23889		DB2 10 support for PLAN_TABLE for component EXPLAIN.	UK61139
PM24082		Various fixes and improvements to be considered part of the GA installation package.	UK61317
PM24083		Updates including columns to ACCOUNTING FILE DDF, GENERAL and PROGRAM.	UK65325
PM32638		Collection of new functionality and development fixes.	UK65399
PM32647		Collection of new functionality and development fixes.	UK65412
PM35049		Collection of new functionality and development fixes.	UK65924
PM47871		Batch Record Trace can now "FILE" IFCID 316 and 401 trace data (dynamic and static SQL statements evicted from caches, IFI READA data) into a DB2 LOAD format.	UK72590
PM52470		RKD2VS0x (archive) files increased from 10 to 20. Near Term History can support up to 60 VSAM files.	UK74519
PM57666		Various abends and threads hung.	UK76424
PM67774		Monitor trace class 29 activates IFCIDs 318 and 400 but causes additional SMF output.	UK81043
PM72029		Storage and CPU usage increases over time when using Near Term History statistics.	UK90634

Abbreviations and acronyms

AC	Autonomic computing	CRCR	Conditional restart control record
ACS	Automatic class selection	CRD	Collect report data
AIX	Advanced Interactive eXecutive from IBM (IBM AIX®)	CRUD	Create, retrieve, update or delete
APAR	Authorized program analysis report	CSA	Common storage area
API	Application programming interface	CSF	Integrated Cryptographic Service Facility
AR	Application requester	CSWL	CONCENTRATE STATEMENTS WITH LITERALS
ARM	Automatic restart manager	CTE	Common table expression
AS	Application server	CTT	Created temporary table
ASCII	American National Standard Code for Information Interchange	CUoD	Capacity Upgrade on Demand
B2B	Business-to-business	DAC	Discretionary access control
BCDS	DFSMShsm backup control data set	DASD	Direct access storage device
BCRS	Business continuity recovery services	DB	Database
BI	Business Intelligence	DB2	Database 2
BLOB	Binary large objects	DBA	Database administrator
BPA	Buffer pool analysis	DBAT	Database access thread
BRF	basic row format	DBCLOB	Double-byte character large object
BSDS	Boot strap data set	DBCS	Double-byte character set
CBU	Capacity BackUp	DBD	Database descriptor
CCA	Channel connection address	DBID	Database identifier
CCA	Client configuration assistant	DBM1	Database master address space
CCP	Collect CPU parallel	DBRM	Database request module
CCSID	Coded character set identifier	DCL	Data control language
CD	Compact disk	DDCS	Distributed database connection services
CDW	Central data warehouse	DDF	Distributed data facility
CF	Coupling facility	DDL	Data definition language
CFCC	Coupling facility control code	DES	Data Encryption Standard
CFRM	Coupling facility resource management	DLL	Dynamic load library manipulation language
CICS	Customer Information Control System	DML	Data manipulation language
CLI	Call level interface	DNS	Domain name server
CLOB	Character large object	DPSI	Data partitioning secondary index
CLP	Command line processor	DRDA	Distributed Relational Data Architecture
CM	conversion mode	DSC	Dynamic statement cache, local or global
CMOS	Complementary metal oxide semiconductor	DSNPARMs	DB2's system configuration parameters
CP	Central processor	DSS	Decision support systems
CPU	Central processing unit	DTT	Declared temporary tables

DWDM	Dense wavelength division multiplexer	ICMF	Integrated coupling migration facility
DWT	Deferred write threshold	ICSF	Integrated Cryptographic Service Facility
EA	Extended addressability	IDE	Integrated development environments
EAI	Enterprise application integration	IFCID	Instrumentation facility component identifier
EAS	Enterprise Application Solution	IFI	Instrumentation Facility Interface
EBCDIC	Extended binary coded decimal interchange code	IFL	Integrated Facility for Linux
ECS	Enhanced catalog sharing	IMS	Information Management System
ECSA	Extended common storage area	IOPR	I/O Request Priority
EDM	Environmental descriptor manager	IPL	initial program load
EJB	Enterprise JavaBean	IPLA	IBM Program Licence Agreement
ELB	Extended long busy	IRD	Intelligent Resource Director
ENFM	enable-new-function mode	IRLM	Internal resource lock manager
ERP	Enterprise resource planning	IRWW	IBM Relational Warehouse Workload
ERP	Error recovery procedure	ISPF	Interactive system productivity facility
ESA	Enterprise Systems Architecture	ISV	Independent software vendor
ESP	Enterprise Solution Package	IT	Information technology
ESS	IBM Enterprise Storage Server®	ITR	Internal throughput rate, a processor time measure, focuses on processor capacity
ETR	External throughput rate, an elapsed time measure, focuses on system capacity	ITSO	International Technical Support Organization
EWLC	Entry Workload License Charges	IU	information unit
EWLM	IBM Enterprise Workload Manager™	IVP	Installation verification process
FIFO	First in first out	J2EE	Java 2 Enterprise Edition
FLA	Fast log apply	JDBC	Java Database Connectivity
FTD	Functional track directory	JFS	Journaled file systems
FTP	File Transfer Program	JNDI	Java Naming and Directory Interface
GB	Gigabyte (1,073,741,824 bytes)	JTA	Java Transaction API
GBP	Group buffer pool	JTS	Java Transaction Service
GDPS®	IBM Geographically Dispersed IBM Parallel Sysplex™	JVM	Java Virtual Machine
GLBA	Gramm-Leach-Bliley Act of 1999	KB	Kilobyte (1,024 bytes)
GRS	Global resource serialization	LCU	Logical Control Unit
GUI	Graphical user interface	LDAP	Lightweight Directory Access Protocol
HALDB	High Availability Large Databases	LOB	Large object
HPJ	High performance Java	LPAR	Logical partition
HTTP	Hypertext Transfer Protocol	LPL	Logical page list
HW	Hardware	LRECL	Logical record length
I/O	Input/output	LRSN	Log record sequence number
IBM	International Business Machines Corporation	LRU	Least recently used
ICF	Internal coupling facility		
ICF	Integrated catalog facility		

LSS	Logical subsystem	RBA	Relative byte address
LUW	Logical unit of work	RBLP	Recovery base log point
LVM	Logical volume manager	RDBMS	Relational database management system
MAC	Mandatory access control	RDS	Relational data system
MB	Megabyte (1,048,576 bytes)	RECFM	Record format
MBps	Megabytes per second	RI	Referential Integrity
MLS	Multi-level security	RID	Record identifier
MQT	Materialized query table	ROI	Return on investment
MTBF	Mean time between failures	RPO	Recovery point objective
MVS	Multiple Virtual Storage	RR	Repeatable read
NALC	New Application License Charge	RRF	Reordered row format
NFM	new-function mode	RRS	Resource recovery services
NFS	Network File System	RRSAF	Resource recovery services attach facility
NPI	Non-partitioning index	RS	Read stability
NPSI	Nonpartitioned secondary index	RTO	Recovery time objective
NVS	Non volatile storage	RTS	Real-time statistics
ODB	Object descriptor in DBD	SAN	Storage area networks
ODBC	Open Database Connectivity	SBCS	Store single byte character set
ODS	Operational Data Store	SCT	Skeleton cursor table
OLE	Object Link Embedded	SCUBA	Self contained underwater breathing apparatus
OLTP	Online transaction processing		
OP	Online performance	SDM	System Data Mover
OSC	Optimizer service center	SDP	Software Development Platform
PAV	Parallel access volume	SLA	Service-level agreement
PCICA	Peripheral Component Interface Cryptographic Accelerator	SMIT	System Management Interface Tool
PCICC	PCI Cryptographic Coprocessor	SOA	Service-oriented architecture
PDS	Partitioned data set	SPL	Selective partition locking
PIB	Parallel index build	SPT	Skeleton plan table
PLSD	page level sequential detection	SQL	Structured Query Language
PPRC	Peer-to-Peer Remote Copy	SQLJ	Structured Query Language for Java
PR/SM	IBM Processor Resource/System Manager (PR/SM™)	SRM	IBM Service Request Manager®
PSID	Pageset identifier	SSL	Secure Sockets Layer
PSP	Preventive service planning	SU	Service Unit
PTF	Program temporary fix	TCO	Total cost of ownership
PUNC	Possibly uncommitted	TPF	Transaction Processing Facility
PWH	Performance Warehouse	UA	Unit Addresses
QA	Quality Assurance	UCB	Unit Control Block
QMF	Query Management Facility	UDB	Universal Database
QoS	Quality of Service	UDF	User-defined functions
QPP	Quality Partnership Program	UDT	User-defined data types
RACF	Resource Access Control Facility	UOW	Unit of work
RAS	Reliability, availability and serviceability	UR	Uncommitted read

UR	Unit of recovery
vCF	Virtual coupling facility
VIPA	Virtual IP Addressing
VLDB	Very large database
VM	Virtual machine
VSE	Virtual Storage Extended

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks publications

The following IBM Redbooks publications provide additional information about the topics in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *DB2 9 for z/OS Performance Topics*, SG24-7473
- ▶ *DB2 10 for z/OS Technical Overview*, SG24-7892
- ▶ *Extremely pureXML in DB2 10 for z/OS*, SG24-7915
- ▶ *LOBs with DB2 for z/OS: Stronger and Faster*, SG24-7270
- ▶ *Effective zSeries Performance Monitoring Using Resource Measurement Facility*, SG24-6645
- ▶ *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01

You can search for, view, or download Redbooks publications, Redpaper publications, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Other publications

These publications are also relevant as further information sources:

- ▶ *Program Directory for DB2 10 for z/OS*, GI10-8829
- ▶ *DB2 10 for z/OS Administration Guide*, SC19-2968
- ▶ *DB2 10 for z/OS Application Programming and SQL Guide*, SC19-2969
- ▶ *DB2 10 for z/OS Application Programming Guide and Reference for Java*, SC19-2970
- ▶ *DB2 10 for z/OS Codes*, GC19-2971
- ▶ *DB2 10 for z/OS Command Reference*, SC19-2972
- ▶ *DB2 10 for z/OS Data Sharing: Planning and Administration*, SC19-2973
- ▶ *DB2 10 for z/OS Installation and Migration Guide*, GC19-2974
- ▶ *DB2 10 for z/OS Internationalization Guide*, SC19-2975
- ▶ *DB2 10 for z/OS Introduction to DB2 for z/OS*, SC19-2976
- ▶ *IRLM Messages and Codes for IMS and DB2 for z/OS*, GC19-2666
- ▶ *DB2 10 for z/OS Managing Performance*, SC19-2978
- ▶ *DB2 10 for z/OS Messages*, GC19-2979
- ▶ *DB2 10 for z/OS ODBC Guide and Reference*, SC19-2980

- ▶ *DB2 10 for z/OS pureXML Guide*, SC19-2981
- ▶ *DB2 10 for z/OS RACF Access Control Module Guide*, SC19-2982
- ▶ *DB2 10 for z/OS SQL Reference*, SC19-2983
- ▶ *DB2 10 for z/OS Utility Guide and Reference*, SC19-2984
- ▶ *DB2 10 for z/OS What's New?*, GC19-2985
- ▶ *DB2 10 for z/OS Diagnosis Guide and Reference*, LY37-3220
- ▶ *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Version 5.1.0, Configuration and Customization*, GH12-6928
- ▶ *IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS Version 5.1.0 Report Reference*, SH12-6921

Online resources

These websites are also relevant as further information sources:

- ▶ DB2 Information Center:
<http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp>
- ▶ DB2 10 for z/OS:
<http://www.ibm.com/software/data/db2/zos/>
- ▶ DB2 Tools for z/OS:
<http://www.ibm.com/software/data/db2imstools/products/db2-zos-tools.html>
- ▶ DB2 for z/OS performance management tools:
<http://www.ibm.com/software/data/db2imstools/solutions/performance-mgmt.html>
- ▶ DB2 Information Management Tools and DB2 10 for z/OS Compatibility:
<https://www.ibm.com/support/docview.wss?uid=swg21409518>
- ▶ The IBM Tivoli OMEGAMON XE for DB2 Performance Expert on z/OS website:
<http://www.ibm.com/software/tivoli/products/omegamon-xe-db2-peex-zos/features.html>
- ▶ The IBM Tivoli OMEGAMON XE for DB2 Performance Expert publications:
http://publib.boulder.ibm.com/infocenter/tivihelp/v15r1/index.jsp?topic=/com.ibm.omegamon.xe_db2.doc/ko2mee1063.htm
- ▶ The technote “Deployment Alternatives - OMEGAMON PE for DB2 on z/OS Extended Insight Analysis Dashboard”:
<http://www.ibm.com/support/docview.wss?uid=swg21456995&myns=swgtiv&mynp=OCSSUSP&mync=R>
- ▶ “IBM System Storage DS8800 Performance” white paper:
Partners: <http://partners.boulder.ibm.com/src/atstr.nsf/WebIndex/WP101799>

Help from IBM

IBM Support and downloads:

ibm.com/support

IBM Global Services:

ibm.com/services

Index

Numerics

64-bit BSAM buffers 71, 284

A

access xxiii, 3–4, 18–20, 71, 78, 95, 141, 149, 154–155, 202, 230–231, 234, 236, 240, 291, 310, 330
access control 9, 292
access path xxiii, 3, 8, 10–11, 19, 115, 121, 155–156, 198–199, 211, 218, 301, 310
active DB2 subsystem 58
administrative authority 4
ALTER BUFFERPOOL 32–33
ALTER TABLE 113, 120, 189, 192, 194, 294
ALTER TABLESPACE 81 command 46
APAR 21–22, 24, 50, 55, 86, 112, 119, 219–220, 230, 288, 293, 308, 327
APPEND=YES 149
application xxiii–xxiv, 1–2, 18, 21, 24, 26, 61, 87, 92, 140–141, 151, 187, 230–231, 234, 240, 270, 292, 319, 330
application developer 330
application environment 187, 220
application period 188
APRETAINDUP 20, 225, 227
asynchronous I/O 56
attribute 213, 271, 331
ATTRIBUTE2 column 338–339
audit categories 292
audit policy 292, 301
authentication 292
AUTHID 335
AUTHORIZATION Id 333
automatic class selection 16
AUX keyword 280
auxiliary index 280
auxiliary table 96

B

base table 4, 26, 78, 108, 189–190, 198, 202, 279–280
BIGINT 186
Binary XML performance 94
bind option 3, 132, 241
BIT 326
BLOB 17
BSDS 283, 308
buffer pool 3, 21, 26, 43, 56, 60, 66, 95, 105, 149, 180, 203, 271–272, 313, 317 activity 32, 60, 272 attribute 32–33
BP0 60, 317
CPU overhead 30

entire size 31
manager 33
page sets 31, 33
scan 180, 212, 280
size 31, 60, 105, 317
space 26, 31, 105, 180, 271
storage 31, 107, 317
support 31, 274
buffer pools 26, 30–31, 50–51, 105, 206, 313, 315
BUSINESS_TIME 5, 188–189
BUSINESS_TIME period 5, 188–189, 197

C

catalog table 17, 20, 110, 220, 292, 308–309
CCSID 331
CF 61
character string 213
CHECK DATA 120, 270, 283
CHECK INDEX 120, 270, 288
CHECK LOB 270, 283
CICS xxiv, 3, 60
class 17–18, 35, 44–45, 54, 86, 166, 168, 232, 281, 293
CLI 94, 131, 133–134, 348
CLI applications 348
CLOB 17, 110
CLUSTER 44, 78, 80, 210
CM 10, 18, 23, 25–26, 33, 260, 283, 303, 308
COLGROUP 277–278
COLLID 335
colon 261
column DRIVETYPE 74
Column name 173, 177
COMMENT 165
components 86, 172, 261, 325
compression 21, 46, 74, 97, 148, 295, 320
compression dictionary 46, 149
condition 66, 158, 188, 331
CONNECT 36, 53, 60, 326
conversion mode 34, 93, 155–156, 206, 256–259, 306
conversion mode (CM) 30, 33, 39, 44–45, 266
COPY 39, 47, 221, 243, 270
COUNT 173, 211, 214, 294
CPU overhead 31, 46, 149, 163, 202, 232–234, 301
CPU reduction 3, 8, 50, 94, 128–130, 205, 240, 289
CPU time 1, 3, 6–7, 18–21, 34, 44, 51–52, 86, 152, 155, 232–233, 238, 297, 311, 327 total cost 44, 168, 174
created global temporary table 37
CS 231, 244
CTHREAD 141
CURRENT EXPLAIN MODE 218–219
CURRENT TIMESTAMP 192
currently committed data 4, 230–231

D

data xxiii, 3–4, 16, 63, 149, 155, 157, 270, 292, 305, 323, 341
data class 16
data page 27, 44, 46–47, 80, 97, 111, 149
data page, different set 80
data set 4, 16, 31, 63, 270, 294, 308
data set, maximum number 141
data sharing xxiv, 7, 21–22, 78, 80, 127, 144, 281, 306–307, 324, 330
Data type 188
database manager address space 56
database system 188
DATASIZE 121
DATE 36, 63, 91, 188, 195, 294, 326
DB2 10 xxiii, 1, 15–18, 49, 123, 134, 140, 153, 187, 230, 239–240, 269–270, 303–304, 323
 access plan stability support 20, 220
 address 4, 6, 20, 33, 50, 134, 143, 220, 325
 authority 4
 base 91, 127
 break 325
 buffer pool enhancements 8, 30
 change 4, 17, 20, 81, 140, 156, 218, 222, 308
 CM9 19, 307
 conversion mode 18, 257, 307
 conversion mode (CM) 30, 33, 39, 44–45, 266
 data 3, 20, 35, 63, 149, 188, 230, 271
 DB2 catalog 4, 17–18, 74, 307
 enhancement 71, 93, 95, 162, 166, 207, 216, 254, 288
 environment 4, 220, 305, 324
 externalize 325, 329
 format 5, 70, 94–95, 286, 308
 function 1, 17, 19, 49, 78, 87, 165, 256–259, 271, 288, 292, 305, 331
 group 307, 329
 index scan 26–27, 156
 load 284
 make 3–4, 32, 178, 210, 307
 memory mapping 325
 new enhancements 334
 new function 2, 9, 87, 91, 306
 new-function mode 34, 46, 149, 305
 NFM 1, 18–19, 46, 91, 149, 188, 307
 page fix 27, 39
 running 3, 141, 260, 281
 SQL 3, 18, 95, 141, 153, 213, 230, 283, 298, 330
 SQL functionality 153
 table spaces 4, 25, 37, 78, 279, 313
 track 203
 use 2, 4, 18, 20, 45, 68, 78–79, 81, 142, 155, 208, 271, 299, 305, 331
 utility 32, 44, 46, 51, 95, 271, 275, 284, 305
DB2 10 for z/OS, performance improvements 2
DB2 8 270, 281
DB2 9 xxiii–xxiv, 2–4, 18–20, 34, 64, 134, 140, 148, 155, 179, 205, 230, 270, 287, 292, 303, 329, 348
DB2 10 3–4, 19–20, 34, 70, 79, 86, 141, 149, 156, 205, 255–256, 305

IPv6 support 260
package 9, 20, 142, 221, 310, 333
partitioned table spaces 78
subsequent REBINDs 20
system 3, 23, 92, 124, 141, 230, 283, 305
DB2 catalog 18
DB2 family 5
DB2 member 46, 126
DB2 startup 24, 39, 63, 293
DB2 subsystem 3–4, 6, 21, 39, 57, 62, 75, 140, 261, 284, 296, 307, 324
DB2 system 6, 95, 126, 181, 189, 306, 334, 338–339
DB2 utility 288–289
DB2 V8 xxiv, 2–3, 8, 20, 63–64, 134, 270, 303
DB2 V8, end of service 304
DB2 Version 1 23, 39
DB2 Version 5 80
DB2 Version 7 325
DB2 Version 8 3, 230
DB2 Version 9 32, 328
db2dsdriver.cfg 351
DBA toolbox 112
DBADM authority 301
DBAT 11, 61, 240, 330, 336
DBD01 21
DBM1 56
DBM1 address space 6, 32, 34, 55–56, 62, 134, 145, 325
DBM1 address space, virtual storage usage 325
DBRM 325
DDF xxiii, 3, 8, 10, 56, 61, 131, 134, 240, 301, 328
DDF command 241
DDF location 324
DDL 10, 17–19, 92–93, 121, 192, 194, 230, 240, 293
DDLTOX 358
death by random I/O 112
DECFLOAT 91
default value 20, 37, 39, 74, 119, 132, 165, 242, 310
deferred write 7, 33–34, 51, 56, 62
DEGREE 10, 162, 244
DELETE 46, 88, 95, 231, 275, 293, 331
delete 21, 39, 78, 235–237
DFSMS 64
DIAGLEVEL1 337
DIAGLEVEL2 337
DISPLAY PROFILE 339
Distributed xxiv, 6, 132–133, 348, 369
distributed data facility (DDF) xxiii, 3, 8, 134
DRDA xxiv, 18, 92, 94–95, 240, 329
DS8000 49, 65, 271
DSN_XMLVALIDATE 92
DSN1COMP 47
DSN1COPY 274
DSN6SPRM 21, 37–38
DSN6SYSP 141
DSNDB01.SYSUTILX 25
DSNJU004 283
DSNSZPARM, SPT01_INLINE_LENGTH 22
DSNT772I 338
DSNTEP2 36, 213
DSNTIJEN job 18, 311

DSNTIPN 74
DSNTSMFD 75, 295
DSNZPARM 37, 44, 74, 141, 231, 233, 271, 283, 310, 324
 ABIND 308
 ACCUMACC 328
 ACCUMUID 328
 COMPRESS_SPT01 22
 DPSEGSZ 80
 EDMPOOL 24
 IMPDSDEF 119
 PLANMGMTSCOPE 20
 PTASKROL 328
 REORG_IGNORE_FREESPACE 354
 SKIPUNCI 230
 SMFCOMP 295
 SPRMRRF 78
 SPT01_INLINE_LENGTH 22
DSNZPARM UTSORTAL=YES 289
DSSIZE 17, 38
duplicate LRSN value 46, 148
Dynamic prefetch 26, 66
dynamic SQL 142, 212, 219, 230, 299, 331
dynamic SQL statement 219
Dynamic statement cache 30

E

efficiency xxiii, 65, 198, 202
element 87–88
ENFM 18, 306
environment 21, 51–52, 57, 62, 80, 126, 130, 151–152, 296, 305, 330, 353
error message 279
EXPLAIN 155, 203, 217–219, 333
expression 96, 164, 201
extended correlation token 260
extended format (EF) 284
extended format data sets 71, 284
extended format, data sets 69
Extended Insight 323

F

FCID 400 332
FCID 401 332
FETCH 35, 96, 121, 158, 224, 331
fetch 164
FICON 43, 65
file reference variables 267, 284
FlashCopy 49, 269–270
FlashCopy enhancements 270, 279, 283
FlashCopy image copy 270
FlashCopy image copy, with COPY utility 271, 274
flexibility 3, 5, 80, 92, 229, 234, 307, 334
function 2–5, 34–35, 44, 46, 75, 78, 127, 165, 193, 238, 269, 271, 304

G

GB bar 124, 134, 145

GENERATED ALWAYS 189
getpage 21, 32, 45, 112–113, 148, 204–205
given table space, XML data 285

H

handle 3, 108, 279
hash access 112
hash key 112
hash locking 116
Hash performance 118
HASH space 120–121
hash space 112, 119
hash table 79, 112
 hash overflow index 115
 space 113, 119
 table space 120–121
hash value lock 116
heavy INSERT, environment 80
history table 189
host variable 89, 176, 207–208, 219, 298
host variables 176

I

I/O xxiii, 16, 26, 51, 95–97, 141, 166, 211–212, 270, 281, 315
I/O parallelism 43–44, 56, 149–150
I/O subsystem 40
IDBACK 141
IDCAMS 16, 271, 274
IDFORE 141
IFCID 124 331
IFCID 148 328
IFCID 172 331
IFCID 196 331
IFCID 225 325
IFCID 225, duplicate data 325
IFCID 3 297, 325
IFCID 316 332
IFCID 318 332
IFCID 337 331
IFCID 350 331
IFCID 359 324
IFCID 365
 record 329
 trace 329
 trace data 329
IFCID 58 331
IFCID 63 331
IFCID 65 331
IFCID 66 331
IFCID 95 36
IFCID 96 36
IFCID record 325
II10817 354
II14219 354
II14334 354
II1440 354
II14426 86, 354
II14438 363

II14441 354
II14464 354
II14474 308, 354
II14477 308, 354
II14564 354
II14587 354
II14619 354
image copy 270
IMMEDWRITE 244
IMPLICIT 216
IMS 3, 127, 130, 304
index xxiii, xxvi, 16, 26, 56, 69, 79–80, 148, 154, 197, 277–278, 324
index ORing 158
index page, split 148
index probing 177
INLINE LENGTH 96
inline LOB 95
Inline LOBs performance 98
INLINE_LOB_LENGTH 96
IN-list predicate 154
inner workfile 164
INSERT 46, 80, 148–149, 177, 231, 293, 331
insert xxiii, 2–3, 6–7, 21, 39, 56, 68, 148, 189, 237
installation 35, 39, 74, 141, 303–304, 337
installation job, DSNTIJSG 334
INSTANCE 36, 326
instrumentation facility interface (IFI) 330
IP address 333, 350
IPADDR 335
IRLM 23, 53, 56, 60–61, 130, 305, 325
IS 54, 59, 62, 218, 225, 242, 288

J

Java 151, 234, 327
JCC 130–131, 212, 300, 341
JDBC 10, 94, 131, 150, 305, 348

K

KB 21, 26, 28, 50, 65–66, 93–94, 148, 179, 284
KB chunk 31
KB page 31, 66, 98, 106
keyword 18, 50, 130, 212, 280, 336

L

LANGUAGE SQL 195
latch contention 6
LIKE 167
limited block fetch 248–249
list xxiii, 26–27, 51, 65, 112, 115, 154, 261, 282, 308, 329
list prefetch xxiii, 26, 56, 69, 155
LOAD 44, 70, 94, 149, 269–270
LOB xxiii, 4, 18–19, 21, 95, 141, 173, 270, 275, 280, 308
LOB column 96
LOB column, inline length 109
LOB table 21, 96, 280
LOB table space 78, 96–97, 280
LOBs xxiii, 17–19, 21, 70, 77, 79, 269, 309

LOBs, processing 95, 286
local Java applications 248
LOCATIONS 329
LOCK 53, 60, 325
locking 10–11, 17–18, 25, 85, 95, 168, 230–231, 309
locks 4, 11, 22, 231, 234, 237, 240, 324
LOG 53, 60–61, 216, 294
log record 23, 39–40
log record sequence number 46
log record sequence number (LRSN) 46
LOGGED 271, 280
logging 39, 41, 43, 46, 150, 280
logging and insert 68
LPAR 19, 27, 130, 166, 206
LRSN 46, 148–149
LRSN spin avoidance 46

M

maintenance 19, 32, 50, 86, 93, 112, 190, 221, 301, 353
management class 16
map page 21, 78, 80
mass delete 78
materialization 89, 181, 254, 259
MAX USERS 141
MAXDBAT 141
maximum number 6, 141
MAXOFILR 141
MB page, frame 31
MEMBER CLUSTER 80
member cluster 44, 80, 82, 150
memory access 3
MERGE 46, 149, 299, 331
Migration 219, 304
MIN STAR JOIN TABLES 333
MODIFY 10, 57, 59, 241, 301, 329
MODIFY DDF 132, 241
monitor class 29 332
MSTR address space 62

N

namespace 87–88
native SQL procedure 8
new-function mode 3–4, 80, 92, 212, 259, 306
DB2 10 4, 80, 306
DB2 9 46, 260, 306
NFM 4, 10, 18–19, 23, 46, 78, 91, 95, 149, 303, 332
node 87
non-data sharing 21
non-LOB 284–285
NPAGES THRESHOLD 333
NPI 51, 278, 281
NULL 172, 189

O

OA03148 362
OA23049 362
OA31116 50, 362
OA32599 362

OA32612 362
 OA33106 137, 362
 OA33529 50, 362
 OA33633 362
 OA33702 50, 362
 OA34865 362
 OA35057 362
 OA35357 362
 OA35885 136, 355, 362
 OA36354 362
 OA37697 362
 OA37821 360, 363
 OA37940 362
 OA38243 363
 OA39087 363
 Object 290
 ODBC 4, 94, 131, 212, 218, 305
 OMEGAMON PE reporter 294
 Optim Performance Manager 332
 optimization xxiii, 2–4, 8, 166, 208, 219, 333
 options 7, 11, 20, 37–38, 105, 212, 219, 231, 241, 271, 292, 310, 348
 ORDER 34–35, 92, 156, 210
 ORDER BY 35, 157, 211
 ORDER BY clause 158
 ordering 159

P

page access 26–27, 112
 page level locking 81
 page set 21, 31–32, 46, 316
 page size 28, 34, 50, 68, 85, 97
 PARAMDEG 165
 parameter marker 213
 PART 154, 281
 PARTITION 47, 175
 partition-by-growth 44, 78–79, 150, 279–280, 309
 partition-by-growth (PBG) 34–35
 partition-by-growth table space 79, 113
 PARTITIONED 130, 210
 partitioned table space 4, 47, 79
 partitioning 78, 112–113, 160, 278
 partitions xxiii, 25, 51, 113, 177, 279
 pending changes 78
 Performance xxv–xxvi, 1–2, 11, 21, 57, 60, 65, 119, 231, 239, 276, 299, 310, 341
 performance xxiii, 1–2, 16–17, 49, 77–78, 80, 143, 153, 187, 231, 269–270, 291, 303, 323
 performance improvement 2, 10, 34, 45, 121, 124–125, 150, 162, 212, 214
 PGFIX 27, 31, 50
 PGSTEAL 32
 physical design 77
 PK28627 354
 PK51045 327
 PK52523 219
 PK56922 308
 PK70060 37
 PK76100 354
 PK80375 21

PK83397 279, 354
 PK83735 149
 PK85856 288
 PK85889 288
 PK90032 92–93
 PK90040 92–93
 PK92339 354
 PKGNAME 335
 PKGREL(COMMIT) 243
 PM00068 64, 354
 PM01821 354
 PM02528 37
 PM04968 308, 321, 354
 PM13466 354
 PM13467 354
 PM13525 355
 PM13631 355
 PM17336 355
 PM17542 64, 355
 PM18196 289–290, 355
 PM18557 64, 355
 PM19034 355
 PM19584 286–287, 355
 PM21277 355
 PM21747 355
 PM22628 363
 PM23887 363
 PM23888 363
 PM23889 363
 PM24082 363
 PM24083 363
 PM24721 21, 355
 PM24723 136, 355
 PM24808 355
 PM24937 355
 PM25271 355
 PM25282 355
 PM25357 355
 PM25525 355
 PM25635 355
 PM25648 356
 PM25652 119, 356
 PM25679 220, 222, 356
 PM26475 356
 PM26480 356
 PM26762 356
 PM26781 356
 PM26973 356
 PM26977 301, 356
 PM27073 22, 356
 PM27099 356
 PM27811 22, 356
 PM27828 356
 PM27835 356
 PM27872 75, 295, 356
 PM27962 356
 PM27973 356
 PM28100 356
 PM28296 293, 301, 356–357
 PM28385 91, 357

PM28458	357
PM28500	357
PM28796	357
PM28925	357
PM29037	112, 357
PM29124	357
PM29900	357
PM30425	357
PM30468	34, 55–56, 62, 357
PM30991	357, 360
PM31003	357
PM31004	357
PM31006	357
PM31009	357
PM31214	357
PM31243	357
PM31313	357
PM31314	357
PM31614	24, 127, 357
PM31641	357
PM31807	357
PM31813	358
PM32638	363
PM32647	363
PM33501	358–359
PM33767	358
PM33991	358
PM35049	363
PM35190	358
PM35284	358
PM36177	358
PM37112	358–359
PM37293	358
PM37300	358
PM37625	358
PM37647	136, 355, 358
PM37660	358
PM37672	358
PM37816	359
PM37956	359
PM38164	359
PM38417	359
PM39342	359
PM40388	359
PM40501	359
PM41447	359
PM42331	359
PM42528	359
PM42560	359
PM42924	359
PM43293	359
PM45318	359
PM45650	359
PM45810	359
PM47871	363
PM48358	360
PM49816	360
PM50140	360
PM51467	360
PM51945	360
PM52470	363
PM52724	357, 360
PM52727	359–360
PM52788	360
PM55051	360
PM55933	360
PM56355	360
PM56363	360
PM56429	360
PM56542	360
PM56725	360
PM56845	360
PM57632	360
PM57666	363
PM58915	360
PM60233	360
PM60236	360
PM60732	360
PM60826	361
PM62709	361
PM62797	361
PM63219	361
PM63505	361
PM63753	361
PM64226	361
PM64230	361
PM64602	361
PM65236	361
PM65360	361
PM65550	361
PM65767	361
PM66173	361
PM66882	361
PM67774	363
PM70046	361
PM70181	361
PM70270	361
PM7114	362
PM72029	363
PM72997	362
PM74659	362
PM76924	362
POSITION	288
PRDID	335
predicate	34, 86, 115, 154, 198, 301
prefetch quantity	26
prefetch quantity, additional requests	26
Private protocol	330
PTF	21, 34, 56, 62, 327
pureXML	4–5, 85
Q	
query	4–5, 18–19, 26, 51, 105, 155, 198, 201, 238, 299, 328
query performance	73, 166, 208, 301
R	
RANDOM	61
range-partitioned table space	79, 113

RBA 23, 39, 279
READS 61, 329, 332
Real 34, 177
real storage 31, 40, 50, 134, 190
real time statistics 121
reason code 337
REBIND PACKAGE 20, 222, 310
RECOVER 39, 120, 270, 305
Redbooks publications website 369
Redbooks Web site
 Contact us xxvi
referential integrity 121, 149, 202–203, 283, 309
REGION 326
RELEASE(DEALLOCATE) 241
remote location 329
REOPT 176, 244
reordered row format 78, 96, 112
REORG 18, 30, 46, 73–74, 81, 85, 130, 270–271, 308
REORG TABLESPACE 281
REORG utility 119, 280
REPEATABLE 275
REPORT 24, 55, 63, 283–284
REPORT RECOVERY utility 283–284
repository 19, 219, 341
requirements 97, 201, 240, 291, 303–304, 342
RESET 301
Resource Measurement Facility™ 57
result sets 254
return 10, 97, 235, 238, 275, 331
RID xxiii, 38, 179
RIDs 26, 158, 179
ROLE 335
ROLLBACK 39, 60
rollup record 328
row length 34
row level locking 81
row level sequential detection 26, 30
ROWID 105
RRF
 see reordered row format 96
RRSAF 328
RTS 21, 47, 119, 121, 149, 177
RUNSTATS 7, 51, 130, 177, 224, 275

S

same page 46
scalar functions 5, 173
schema 2–5, 78, 86, 269, 277
SECADM authority 293, 301
SECMINT category 293
SECP 327
secure access 291
segmented table space 38–39, 78–79
SEGSIZE 79
SEQ 53, 60, 272
Server xxiv, 94, 124, 341
SET 3, 8, 87, 192, 218, 242, 260
SHRLEVEL 44, 46, 112, 120, 149, 270
side 165, 212, 231, 348
simple table space 79

SKIP LOCKED DATA 230
skip-level migration 304
SMF 49, 53, 57, 294, 324
SNA 260
sort key 34
sort key length 34
sort record 34–35, 288
 row length 34
 sort key length 34
space map 21, 80, 202
space map page 21, 80
sparse index 35, 164
spin avoidance 46
SPT01 20–21, 226, 310
SPUFI 60, 217
SQL xxiii–xxiv, 3–5, 18, 26, 85, 141, 149, 153, 187, 230, 261, 293
SQL PL 5
SQL procedure 5
SQL procedures 260
SQL statement 34, 89, 141, 149, 154, 198, 212, 214, 298, 331
 DISTINCT specifications 34
 original source CCSID 331
SQL stored procedure 20
SQL stored procedures 132
SQL table 5
SQL/XML 85
SQLCODE 34, 37, 112, 218–219, 337
SQLCODE -30041 338
SQLERROR 220, 244
SQLJ 94, 131, 305
SQLSTATE 34, 112
STAR JOIN 333
START PROFILE 339
statement 18, 34, 80, 87, 135, 149, 154, 230, 243, 261, 271, 274, 330
static SQL xxiii, 142, 219, 230, 294, 330
 last bind time 331
static SQL, last bind time 331
STATIME 329
statistics 2, 24, 35, 47, 51, 57, 121, 130, 160, 204, 219, 275, 295, 309, 324
STATUS 309, 335
STMT_ID 298, 331
STMT_ID column value 331
STMTID 332
STOP PROFILE 339
storage class 16
STORGRP 16
straw model 162
SUBSTR 97, 173, 300
synchronous I/O 26, 40–41, 43, 45, 106, 149, 212, 315
SYSADM 36, 299
SYSCOLUMNS 110
SYSIBM.DSN_PROFILE_ATTRIBUTES 333
SYSIBM.DSN_PROFILE_ATTRIBUTES_HISTORY 333
SYSIBM.DSN_PROFILE_HISTORY 333
SYSIBM.DSN_PROFILE_TABLE 333
SYSIBM.DSN_STATEMENT_RUNTIME_INFO 333

SYSIBM.SYSAUDITPOLICIES	294
SYSIBM.SYSCOPY	283
SYSIBM.SYSINDEXSPACESTATS	74, 121
SYSIBM.SYSPACKSTMT	331
SYSIBM.SYSTABLESPACE	78
SYSIBM.SYSTABLESPACESTATS	74, 121
SYSIN	54, 59, 63, 217, 325
SYSLGRNX	279
SYSPACKAGE	20
SYSPACKCOPY	20
SYSPACKSTMT	261
SYSPRINT	59
SYSREC data set	285
SYSTABLEPART	119
System DBADM	301
system parameter	177, 333
system period	188
System z	xxiv, xxvi, 3, 6, 19, 43, 49, 124, 327
System z9	50, 327
SYSTEM_TIME	5, 188–189
SYSTEM_TIME period	5, 188
T	
table expression	165
table row	45
table space	
buffer pool	31, 106, 280
CHKP status	283
data	4, 46, 73, 81, 150, 275, 279
data pages	31
data set	69
definition	280
execution	18, 26, 275
level	269
name	25
option	47, 80, 269
organization	77–78
page	4, 31, 96, 149
partition	4, 46
scan	58, 159
structure	18, 79
type	37, 73, 79
type conversion	79
table space scan	58
tables	2, 5, 18–19, 44, 55, 74, 79, 155, 188, 207, 270, 279, 296, 308, 333
TABLESPACE statement	80, 279
TCP/IP	61, 260, 308, 336
temporal	2, 5, 7, 188
TEXT	224, 354, 362–363
TIME	53, 56, 60, 188, 272, 326–327
times DB2	121, 217
TIMESTAMP	91, 188–189
timestamp column	207
timestamp data	207
timestamp precision	207
TIMESTAMP WITH TIME ZONE	206–207
TIMESTAMP WITHOUT TIME ZONE	188, 207
TOTALROWS	121
trace record	297, 324
traces	53, 58, 75, 293, 301, 323, 330
transitive closure	156
triggers	61, 188, 190, 254, 329
TYPE	36–37, 59–60, 78, 221, 288
U	
UA07148	362
UA38770	362
UA55970	362
UA56174	362
UA56176	362
UA56302	362
UA56642	362
UA57243	362
UA57254	362
UA57704	362
UA58937	362
UA60368	362
UA60823	362
UA64823	363
UA65482	363
UA65808	362
UDF	23, 53, 60–61
UK37397	354
UK50932	354
UK51679	354
UK53480	354
UK58204	354
UK59887	355
UK60466	355
UK60887	355
UK61093	363
UK61094	363
UK61139	363
UK61142	363
UK61213	355
UK61317	363
UK61694	362
UK62150	355
UK62201	355
UK62326	354
UK62328	354
UK63087	355
UK63215	355
UK63366	356
UK63457	21, 355
UK63818	356
UK63820	356
UK63890	357
UK63971	355
UK64370	355
UK64389	356
UK64423	34, 56, 62, 357
UK64424	356
UK64588	355
UK64597	356
UK65205	356
UK65253	357
UK65325	363
UK65332	356

UK65379 22, 356
UK65385 356
UK65399 363
UK65412 363
UK65632 356
UK65637 357
UK65750 357
UK65920 357
UK65924 363
UK65951 356
UK65969 355
UK66046 357
UK66087 355
UK66136 357
UK66327 357
UK66374 24, 127, 357
UK66376 357
UK66379 356
UK66475 355
UK66476 357
UK66610 356
UK66964 357
UK67132 357
UK67267 355
UK67292 356
UK67512 357
UK67578 357
UK67634 358
UK67637 357
UK67639 358
UK67958 357
UK68097 355
UK68098 358
UK68364 357
UK68476 355
UK68652 355
UK68659 358
UK68743 358
UK68801 359
UK69029 358
UK69030 358
UK69055 358
UK69058 358
UK69199 359
UK69286 357
UK69377 358
UK69494 358
UK69607 355
UK69735 358
UK69784 359
UK70215 357
UK70233 356
UK70302 357
UK70310 356
UK70426 359
UK70483 359
UK70513 359
UK70647 356
UK70844 359
UK71128 358–359
UK71333 359
UK71412 357
UK71419 356
UK71420 359
UK71875 359
UK72557 359
UK72590 363
UK73426 359
UK73478 358
UK74175 359
UK74381 360
UK74519 363
UK74556 360
UK74760 360
UK74840 360, 363
UK75324 360
UK75499 360
UK76060 360
UK76344 360
UK76424 363
UK76645 360
UK77001 360
UK77298 360
UK77343 360
UK77500 360
UK77584 359
UK77918 360
UK78231 360
UK78390 360
UK78632 360
UK78678 360
UK78910 361
UK79281 361
UK79368 361
UK79550 361
UK79701 361
UK79898 361
UK80107 361
UK80113 360
UK80191 361
UK80522 361
UK80552 361
UK80678 361
UK81043 363
UK81045 361
UK81047 361
UK81340 361
UK81719 361
UK82555 361
UK82787 360
UK83168 361
UK83171 362
UK83215 361
UK83533 362
UK90197 362
UK90325 359
UK90618 360
UK90634 363
UNIQUE 210
unique index 121, 203, 208

unique index, additional non-key columns 210
universal table space 77, 81, 91, 113, 149–150
universal table space (UTS) 44, 80, 269
UNLOAD 47, 71, 94, 284
UPDATE 31, 46, 53, 60, 80, 87, 127, 190, 267, 275, 293, 308, 331
USE CURRENTLY COMMITTED 230
use PLANMGMT 20
user-defined function 92
UTF-8 351
UTS 78, 80, 269, 287
UTSERIAL lock 25

Z
z/Architecture 65
z/OS xxiii
z/OS 1.10 50, 71, 288
z/OS 1.11 64, 130, 190, 246, 272, 288
z/OS 1.12 51, 57, 63–64, 272, 305
z/OS 1.9 71
z/OS enhancement 284
z/OS Installation 304
zAAP 190, 327
zIIP 7, 27, 30, 44, 50–51, 92, 190, 278, 324

V

VALUE 183
VALUES 177, 191–192
VARCHAR 91–92
variable 35, 89, 173, 207, 288, 298
Version xxiv, 3, 5, 23, 39, 61, 80, 155, 230, 239, 294, 305, 325–326, 370
versioning 5, 78, 188–189
versions 5, 10, 18, 31–32, 81, 91, 135, 156, 188, 201, 230, 310
virtual storage
 consumption 266
 relief 4, 6, 134
 use 6, 145, 165
virtual storage relief 143
VPSIZE 31

W

well-formed XML 93
WFDBSEP 37
WITH 47, 58, 217, 230, 234, 241, 261, 338
WITH RETURN TO CLIENT 254, 259
WLM 23, 31, 34, 53, 56, 60, 92, 240, 330
work file xxiii, 34, 164, 179
workfile 34–35, 37, 159
WORKFILE database 37
workfile record 34
workfile table space 34, 37
workfiles 34–35, 37

X

XML 3, 5, 19, 46, 61, 77, 173, 216, 269, 284, 327
XML column 91–92
XML columns 19, 91, 284
XML data 46, 85
XML data type 85, 93
XML schema 5, 92
XML schema validation 92
XML schema validation performance 92
XML type modifier 92–93
XMLMODIFY function 87
XMLPARSE 92
XPath 86



DB2 10 for z/OS Performance Topics

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



DB2 10 for z/OS Performance Topics



Discover the functions that provide reduced CPU time in CM and NFM

Understand improved scalability and availability

Evaluate the impact of new functions

DB2 10 can reduce the total DB2 CPU demand from 5-20% when you take advantage of all the enhancements. Many CPU reductions are built in directly to DB2, requiring no application changes. Some enhancements are implemented through normal DB2 activities through rebinding, restructuring database definitions, improving applications, and utility processing. The CPU demand reduction features have the potential to provide significant total cost of ownership savings based on the application mix and transaction types.

Improvements in optimization reduce costs by processing SQL automatically with more efficient data access paths. Improvements through a range-list index scan access method, list prefetch for IN-list, more parallelism for select and index insert processing, better work file usage, better record identifier (RID) pool overflow management, improved sequential detection, faster log I/O, access path certainty evaluation for static SQL, and improved distributed data facility (DDF) transaction flow all provide more efficiency without changes to applications. These enhancements can reduce total CPU enterprise costs because of improved efficiency in the DB2 10 for z/OS.

DB2 10 includes numerous performance enhancements for Large Objects (LOBs) that save disk space for small LOBs and that provide dramatically better performance for LOB retrieval, inserts, load, and import/export using DB2 utilities. DB210 can also more effectively REORG partitions that contain LOBs.

This IBM Redbooks publication provides an overview of the performance impact of DB2 10 for z/OS discussing the overall performance and possible impacts when moving from version to version. We include performance measurements that were made in the laboratory and provide some estimates. Keep in mind that your results are likely to vary, as the conditions and work will differ.

In this book, we assume that you are somewhat familiar with DB2 10 for z/OS. See DB2 10 for z/OS Technical Overview, SG24-7892-00, for an introduction to the new functions.

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks